Design of an ATM Switch and Implementation of Output Scheduler

Jun Fang, B.Sc.

Thesis submitted for the degree of

Master of Engineer Science

in the department of Electrical and Electronic Engineer

The University of Adelaide

Adelaide, Australia

15 March 1999

Contents

Abstract VII

Declaration VIII

Acknowledgments IX

List of Figures X

List of Tables XII

Chapter 1 Introduction 1

1.1 Trends in the development of telecommunication network 1

1.2 The advantage of ATM 1

1.3 ATM switching system 2

1.3.1 Switch matrix 3

1.3.2 Port controller 8

1.3.3 Multi-stage switching 9

1.4 A time scheduling ATM switch 10

1.4.1 Structure 10

1.4.2 Input port controller and header processor 12

1.4.3 Scheduler 12

1.5 Description of a Time scheduling Algorithm 13

1.5.1 Basic algorithm 14

1.5.2Enhancement 18

1.5.2.1 Priority 18

1.5.2.2 Multicasting 19

1.6 Summary 19

Chapter 2 The Design of ATM switch 20

2.1 Overview of the ATM switch 20

2.1.1 Design Objective 20

2.1.2 Switching process 21

2.1.3 Interface definition 23

2.2 Switch Matrix 25

2.3 Design of the input port controller 28

2.3.1 Overview of the input port controller 28

2.3.2 The function of the header processor 30

2.3.3 The operation of the buffers 30

2.3.3.1 Temporary buffers 31

2.3.3.2 The structure of the main buffers 32

2.3.3.3 Operation of the main buffers. 33

2.4 Design of the output scheduler. 40

2.4.1 The structure of the output scheduler. 42

2.4.2 Operation of the output scheduler. 43

2.4.2.1 Data flow of the output scheduler. 43

2.4.2.2 Structure analysis. 50

2.4.2.3 Defeating unfairness. 51

2.5 Summary. 53

Chapter 3 Investigating Basic Functional Blocks of the Output Scheduler 54

3.1 Elementary scheduler 54

3.1.1 Structure 54

3.1.2 Operation of the comparison unit 56

3.1.3 Modifications for Priority. 58

3.1.4 Operation of the schedule register. 59

3.1.5 Input address generation. 61

3.1.6 Circuit design of the comparison unit. 64

3.1.6.1 Generating the schedule. 64

3.1.6.2 Updating the output status. 67

3.1.6.3 Interfacing with subsequent units. 71

3.2 Input status register. 73

3.2.1 The structure of the input status register. 73

3.2.2 Operation of the input status register. 75

3.3 Output status register. 77

3.4 Clock generator. 77

3.5 Summary. 82

Chapter 4

Physical Design of the Output Scheduler **83**

4.1 The design methodology. 83

4.2 Techniques for high performance digital design. 86

4.2.1 Design specification. 86

4.2.2 Design requirements 87

4.2.3 Design techniques for high speed. 88

4.2.3.1 Floorplanning. 88

4.2.3.2 Clock distribution and skew. 91

4.2.3.3 Critical path analysis and optimisation. 92

4.2.3.4General techniques to decrease delay 95

4.2.4 Design techniques for low power dissipation. 101

4.2.4.1 Logic family selection. 101

4.2.4.2 Reducing the effective capacitance. 102

4.2.5 Techniques for reducing the area. 106

4.2.6 I/O system Design. 107

4.2.7 Power distribution. 112

4.3 The simulation result of output scheduler. 114

4.3.1 Delay and power dissipation. 114

4.3.1.1 Simulation environment. 114

4.3.1.2 Selection of the stimuli. 115

4.3.1.3 Simulation result. 116

4.3.2 Size and area 118

4.4 Summary. 118

Chapter 5 Discussion **120**

5.1 Speed-up two. 120

5.2 A possible way to improve the speed. 124

5.3 Challenges on packaging. 127

5.4 Summary. 129

Chapter 6 Conclusion **131**

Reference **133**

Abstract

ATM (Asynchronous Transfer Mode) is regarded as the solution for next generation telecommunication network. ATM switches are the critical parts of an ATM network. In this project, an experimental 16 x 16 input buffered ATM switch is developed, which employs a time scheduling algorithm developed by Sarkies and Main. The ATM switch discussed in this thesis contains four basic functional blocks: switch matrix, output scheduler, input port controller and output port controller. The output scheduler is the key part of the project, which is designed to the chip-level. The other parts of the switching system that interface with the output scheduler are designed to the architecture level. The objective of this research project is not only to design a high-speed output scheduler that can support the switch matrix working at 10Gb/s/channel, but also to design the output scheduler that can provide variable priority threshold and multicast to improve the performance.

The output scheduler described in this thesis is designed with TSMC 0.25μm CMOS technology. The entire chip contains over 600,000 transistors. The simulation results show that delay of the critical path is 20.8ns, which is much less than the design requirements, 40ns. The estimated power dissipation of the circuits is 0.785w with a 2.5v power supply at 100°C. The circuit area is 4.9 mm^2 and chip area is about 12mm^2 .

We demonstrate the output scheduler can coordinate with the other parts in the ATM switch to provide high quality service.

Declaration

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the test.

I give consent to this copy of my thesis made, when deposited in the University Library, being available for loan and photocopying.

SIGNED: .

DATE: ... 20/08/99

Acknowledgments

I would like to express my appreciation to the people who made their contributions to the completion of this research. First of all, I would like to thank my supervisor, Dr. Kenneth Sarkies. He not only gave me invaluable guidance and continued support in the course of my research, but also gave me many comments on the structure, contents and grammar of my thesis. Secondly, I also wish to extend my sincere gratitude to Mr Kiet N. To. He patiently taught me the usage of the CAD tools used in VLSI design. Last but not least my thanks go to Mr Andrew Beaumont-Smith, Mr Said Al-Sarawi, Dr. Alireza Moini and Mr. Michael Liebelt for many discussions in various stages of design.

List of Figures

1 A general model for ATM switch 3

2 Architecture of crosspoint switch developed by Lowe 5

3 Architecture of crosspoint switch developed by Savara and Turudic 7

4 Structure of an input buffered ATM switch 11

5 Diagram to illustrate the time scheduling algorithm 15

6 Flow chart of operation switching process 16

7 High-level architecture of an input buffered ATM switch 22

8 Architecture of crosspoint switch used in this ATM switch 26

9 Diagram of input port controller 29

10 Structure of write controller 36

11 Operation of the write controller 39

12 Structure of output scheduler 41

13 Detailed structure of output scheduler 44

14 Flow chart of operation of output scheduler 45

15 Diagram of the elementary scheduler 55

16 A column of elementary scheduler 62

17 Circuit of comparison unit 65

18 Diagram of input status register 74

19 Diagram of output status register 78

20 Diagram of clock generator 79

21 Timing diagram of signal generator 81

22 Floorplan of elementary scheduler 90

23 Structure of buffer distribution 93
24 The critical path of output scheduler 96
25 Circuit of an NAND gate 99
26 Avoid extensive bus sharing 105
27 Placement of pads 108
28 Transient current wave form 110
29 Simulation of critical path 117
30 Architecture of an input buffered ATM switch with speed-up two 121
31 Diagram for an elementary scheduler group 123
32 Structure and circuit of elementary scheduler with lookahead 126

List of Figure

1. Truth table of schedule generation 67

2. Truth table of output status updating 69

3. Truth table of generating fairness signal 70

4. Truth table of generating interface signal 74

Chapter One Introduction

1.1 Trends in the Development of Telecommunication Network

Nowadays, computers have infiltrated all walks of life, such as the home, banks, manufacturing industries and so on. Although stand-alone computers are still widely used, in more and more instances they are networked. As a result, both computer and telecommunication networks are developing rapidly.

Recently, the telecommunication network has acquired two new characteristics: one is the need to support services of different characteristics, for example digitised video and image; the other is the need to support all of these services on a single network [1]. Hence, a new telecommunication network standard is being established, namely B-ISDN (Broadband Integrated Services Digital Network). A great deal of research work is taking place to find a solution for the B-ISDN. Eventually, ATM (Asynchronous Transfer Mode) was agreed as the *target transfer mode* for implementing a B-ISDN [2].

1.2 The Advantage of ATM

ATM is a type of packet switching system that operates at high speed. In an ATM network, the information is transferred asynchronously in the form of cells, a type of small, fixed length packet. Cells have a length of 53 octets and are comprised of two

parts: one is the header that contains the routing information, the other is the information field that is the cell payload. On the sending end the information is organised into cells, on the receiving end the information in the cells is recombined together. Such a technique provides great flexibility, so that the ATM network is very suitable for new high-bit-rate services. Real-time video is a typical example for this kind of service, which is known as variable bit-rate (VBR) service. It is intuitively obvious that the bit rate generated by a video picture of static scenery is certainly different from that of a racing fox. When the scene switches from the scenery to the fox, there is a large burst of information to be transferred. ATM can allow this variable rate of information generation to be transferred effectively across the network. Moreover, another inherent advantage of ATM network is multiplexing of cells. Cells from different services can be transferred onto one link. That means the network operator need only provide one connection to the customers and all services can be provided over this link. Also, as described previously, a significant characteristic of modern telecommunication networks is to support multimedia services. ATM networks use a standard size of cell for all media, which means that switching of the cell streams can be performed at very high rate and this simplifies the design of the ATM switch considerably. Clearly, ATM offers great ease of integration of sources. It was for this reason that ATM was selected as the transfer mode for the new generation of high-speed telecommunication network.

1.3 ATM Switching system

Generally, an ATM switching system is comprised of three elements: switch matrix, an input port controller (IPC) for each input port and an output port controller (OPC)

for each output port [3]. A diagram that shows these three modules is sketched in figure1.

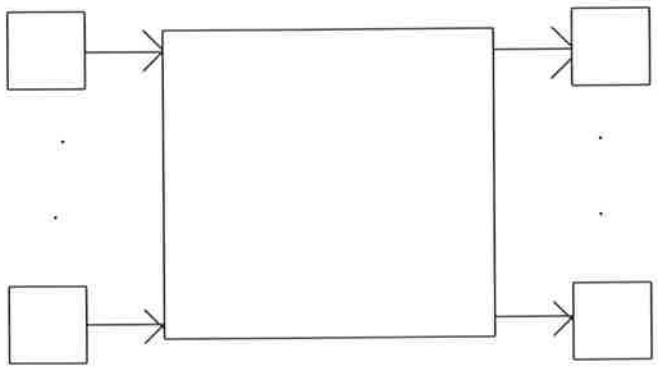

Figure 1: A general model for ATM switch

1.3.1 Switch Matrix

Clearly, the switch matrix is the core of this switching system, in which the data path is developed between any input-output-pair. Hence, matrix will significantly affect the capability of the whole switch. The increasing traffic and multiple services in the modern telecommunication network raise the requirement for high-capability switches. High-performance switches should present such characteristics as very high speed, versatility of switching mode (selective and broadcast), ease of control, small loss, small delay, good signal integrity with little noise and so forth. Furthermore, for the ATM network the capability of handling asynchronous data is of importance. The crosspoint switch has proven to be a competent candidate to meet these rigorous requirements.

In the past few years, much work has been done to implement the high-speed crosspoint switches that can support ATM switch. We shall describe two different architectures used in giga-bit crosspoint switches.

In 1997, Lowe reported a 10Gb/s/channel crosspoint switch [4], which employs a multiplexer/decoder type of architecture. The architecture of this crosspoint switch is shown in figure 2. As shown in the diagram, the switch consists of 16 selector slices. In each selector, there is a 4-bit register for the input address, a 16:1 multiplexer and an output buffer. The 4:16 decoder selects the desired output among the 16 selector slices according to the output address. In each input port there are sixteen input data buffers and input address buffers. They are used to drive the cells and addresses to all the selector slices.

When a cell arrives at the switch, its input address and output address are sent to the input address buffers and decoder respectively. According to the output address the decoder selects one of the 16 outputs, namely one of 16 selector slices. Note that the decoded output address is latched by a load pulse. That means that only when the load pulse is asserted can the output address enter the selector slice. On the other hand, the input address is buffered with low impedance amplifiers and driven to all the slices. The cell is also broadcast to all the slices. Referring to the detailed structure of each slice, we note that the input address is stored in a register and it will not be sent to the multiplexer until it receives a clock signal from the decoder. When the load pulse is asserted, the output address is loaded into each selector slice and the

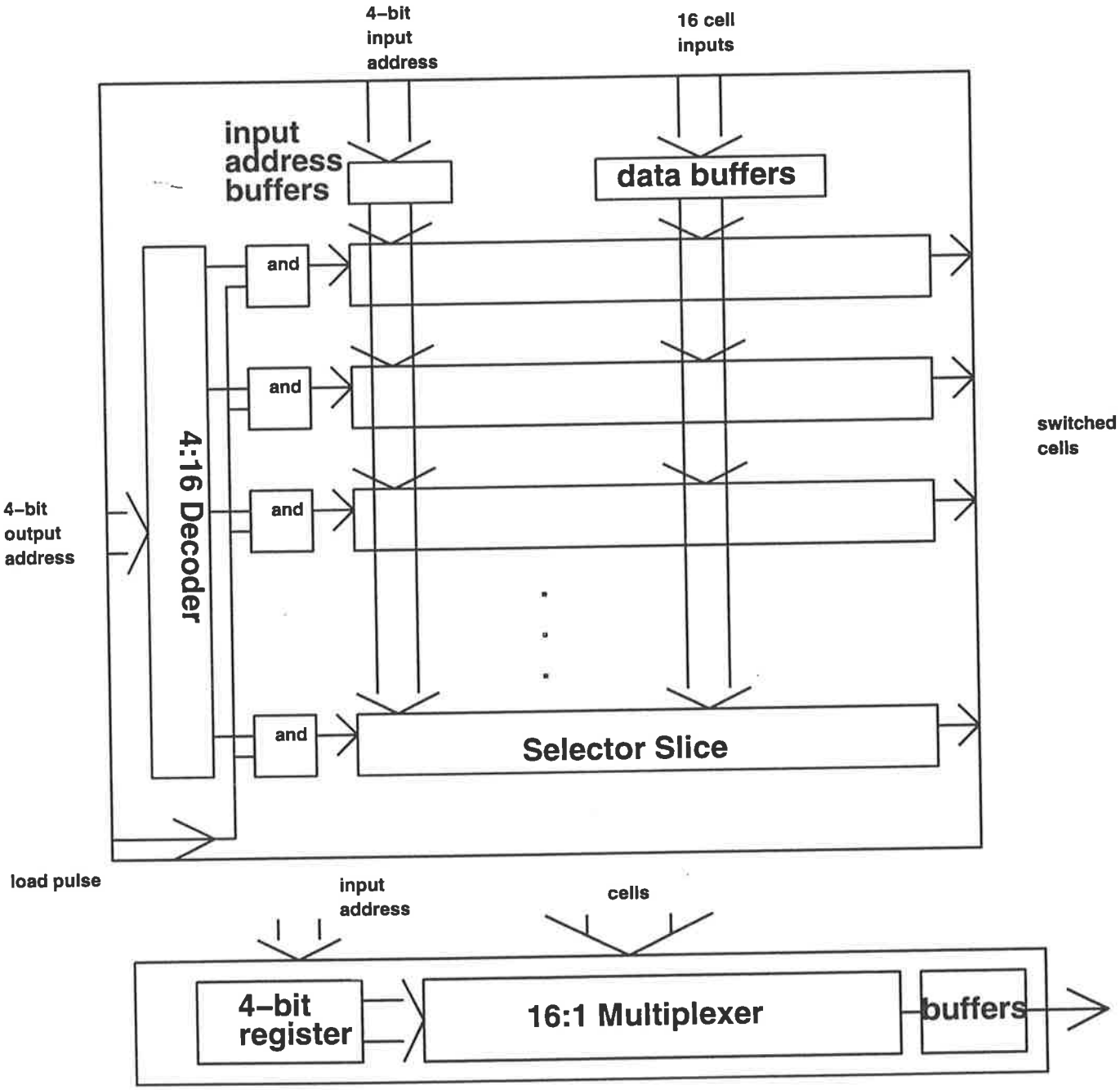

Figure 2 Architecture of crosspoint switch developed by Lowe

asserted bit will activate the register to send the input address to the multiplexer. According to this address, the multiplexer connects one of the cells to the buffer. Finally, the buffers drive the cell out of the switch.

This broadcast-and-select architecture needs a relatively simple structure, so it can be implemented in one chip. This characteristic is significant as it improves the speed of the switch and decreases the cost of production. However, this architecture also has some disadvantages. This switch can set up only one data path at one time, so it needs 16 consecutive load pulse to fully reconfigure the whole switch. That means in one time slot the switch must be programmed 16 times. This increases the complexity of external control and the complexity of interactions with the data flow.

Savara and Turudic developed another architecture for a crosspoint switch in 1995[5] (see figure 3). As normal switch, it has input buffers, output buffers and a crosspoint switch matrix. There are only sixteen 16:1 multiplexers in the switch matrix and each multiplexer corresponds to one output port. A characteristic of this switch is the utilisation of configuration latches and sixteen 4-bit shift registers to deal with the input addresses.

The incoming cells are stored in the input buffers, and the 4-bit input addresses are sent to the address registers. Each address register corresponds to one multiplexer. The addresses are shifted into registers serially. When all the sixteen 4-bit binary numbers are stored into the shift registers, the switch control centre sends a signal to turn on the configuration latch. Each shift register loads the input address into the

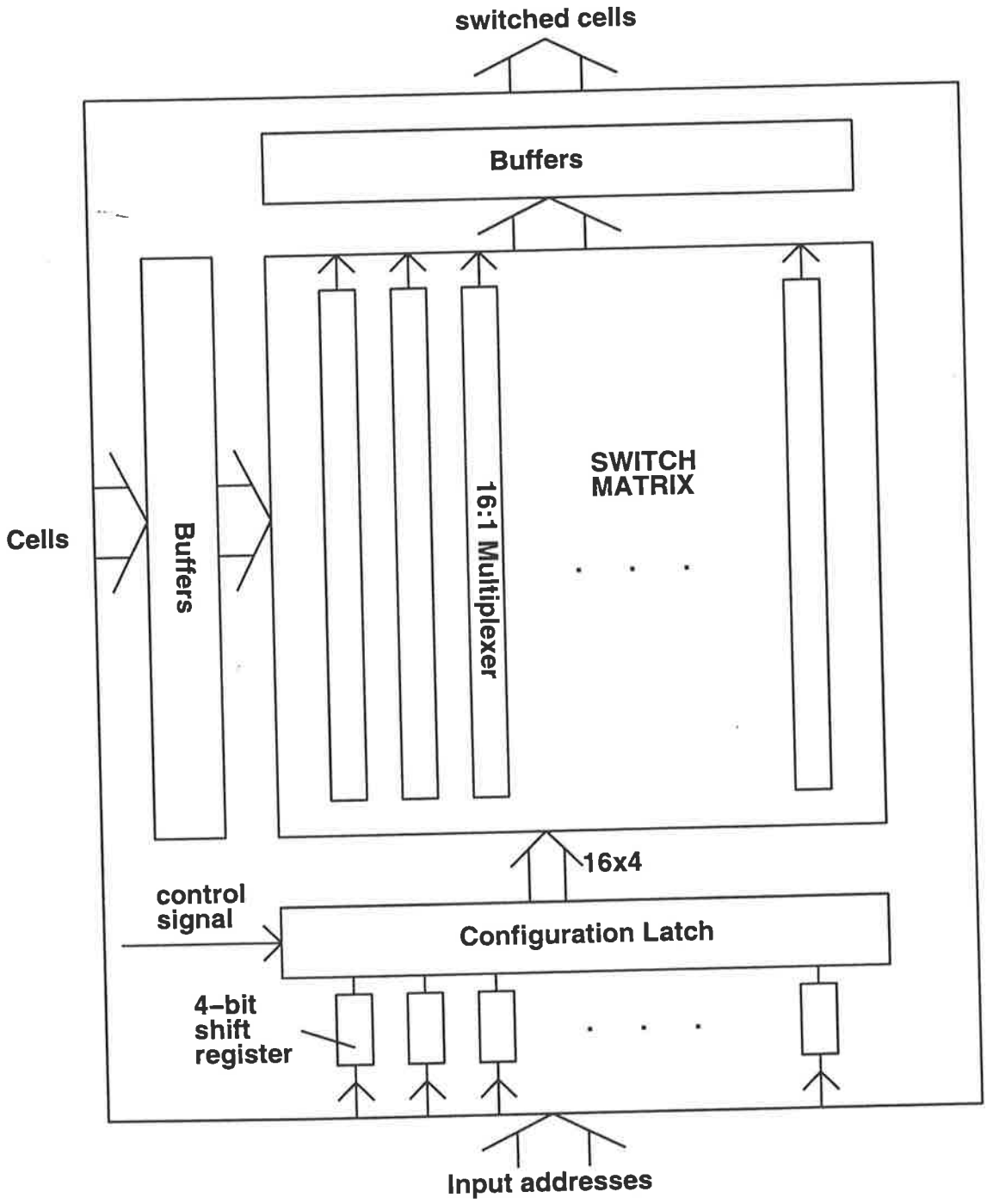

Figure 3 Architecture of crosspoint switch developed by Savara and Turudic

multiplexer in parallel. With the input addresses, the multiplexer will select one of the 16 inputs and connect it to the output. Thus, 16 nonblocking data paths are established at the same time in the switch matrix.

Clearly this architecture is more efficient than the last one, as the cells that asynchronously arrive at the 16 input ports within one time slot are transferred through the switch simultaneously. The switch is configured only once per time slot, which makes its control easy. Hence, this architecture presents a significant advantage when used in an ATM network.

1.3.2 Port Controller

In order to avoid excessive cell loss in the case of internal collisions, buffers have to be provided in the switching system. Basically, there are two possibilities for the buffer location:

- located in the output port controllers
- located in the input port controllers.

Different buffer location of buffers results in different performance.

For the output buffered ATM switch, if the switch matrix can not work fast enough contention may occur. In this case several cells are requesting the same output port simultaneously. In order to achieve collision free switch, the speed-up factor of N must be reached for an N x N switch matrix. That is, it must be possible for each output to pass N cells simultaneously to the buffers. Thus very high speed buffers are

needed. This characteristic makes the output buffered ATM switch undesirable from a performance viewpoint.

The input buffered switch would not suffer from the speed limitations of the output buffered switch, but it has its own problems as well. When first-in-first-out buffers are used in the input port controller, a collision occurs when two or more head-of-the-line cells compete for the same output simultaneously. If so, one cell passes and the others are blocked. All cells in the blocked queues will be blocked, even if they are requesting other possibly unused outputs. Consequently the throughput of the input buffered switch is comparatively low. In order to overcome this disadvantage, some controller module or scheduler must be employed to manage the input queues intelligently. Thus, the input buffered switch is more practical but harder to control.

1.3.3 Multi-stage Switching

For a large switching system, a single stage switch can not provide enough inputs and outputs, so a multi-stage network is used. A multi-stage network is built of several stages which are interconnected by internal links in such a way that any output can be reached from any input. According to the number of paths which are available for a cell to reach a destination output from a given input, these networks can be subdivided into two groups: single-path and multiple-path networks. For the single-path network, due to the fact that only one path exists from an input to an output, routing is very simple. The disadvantage is that when an internal link is used by two inputs simultaneously internal blocking is inevitable. For the multi-path network, due to the existence of alternative paths for reaching the destination output

from a given input, internal blocking can be reduced or avoided. However, the possibility exist for cell streams to arrive at the outputs with cells placed out of sequence.

1.4 A Time Scheduling ATM Switch

In this thesis, we will discuss an input buffered ATM switch using a time scheduling algorithm.

1.4.1 Structure

As stated above, for a high-performance input buffered ATM switch, a controller or scheduler should be employed to manage the queuing. As shown in the figure 4, such an ATM switch consists of these four parts:

1 **The switch matrix** is the core of a switch. The data path between the input and output is set up in it;

2 **The scheduler** selects a suitable time slot in which both the input and output are available to send the cells so that no conflict could occur within the switch matrix;

3 **The input port controller** is the interface between the switch matrix and the scheduler. The controller should coordinate with the scheduler to manage the cell flow. For an input buffered switch, the data buffers are used here to store the incoming cell temporarily;

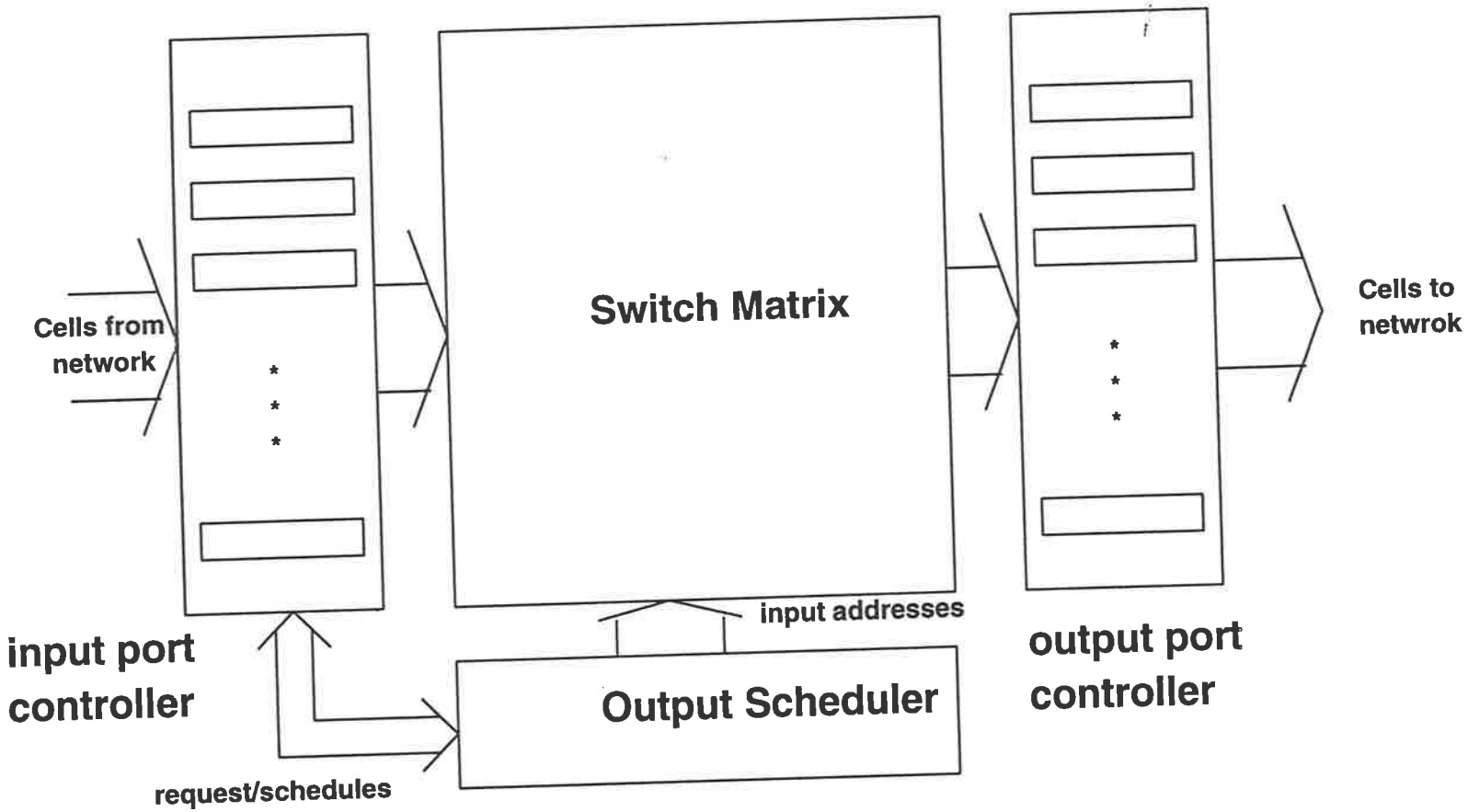

Figure 4 Structure of an input buffered ATM switch

1.5.1 Basic Algorithm

A block diagram is shown in figure 5 to illustrate this algorithm. The basic idea of this algorithm is that in the input port controller and output port controller an input status array and an output status array are maintained and updated respectively. These indicate the usage of the input ports and output ports in successive time slots. Specifically, a status array records which time slots have been scheduled to send a cell for this port and which slots are available for new cell. Referring to figure 5, here we take 7 time slots as an example. Both the input status array and the output status array are maintained in the form of binary number array. In these arrays, *1* represents a time slot that has been scheduled out, while *0* represents a time slot that is available for scheduling. When a cell is to be switched from an input to an output, the corresponding input and output port controller will send the status arrays to the scheduler. The scheduler compares these two status arrays so as to find the first time slot that both input port and output port are available. Finally, the scheduler sends the scheduling results to the input port controller and output port controller, respectively.

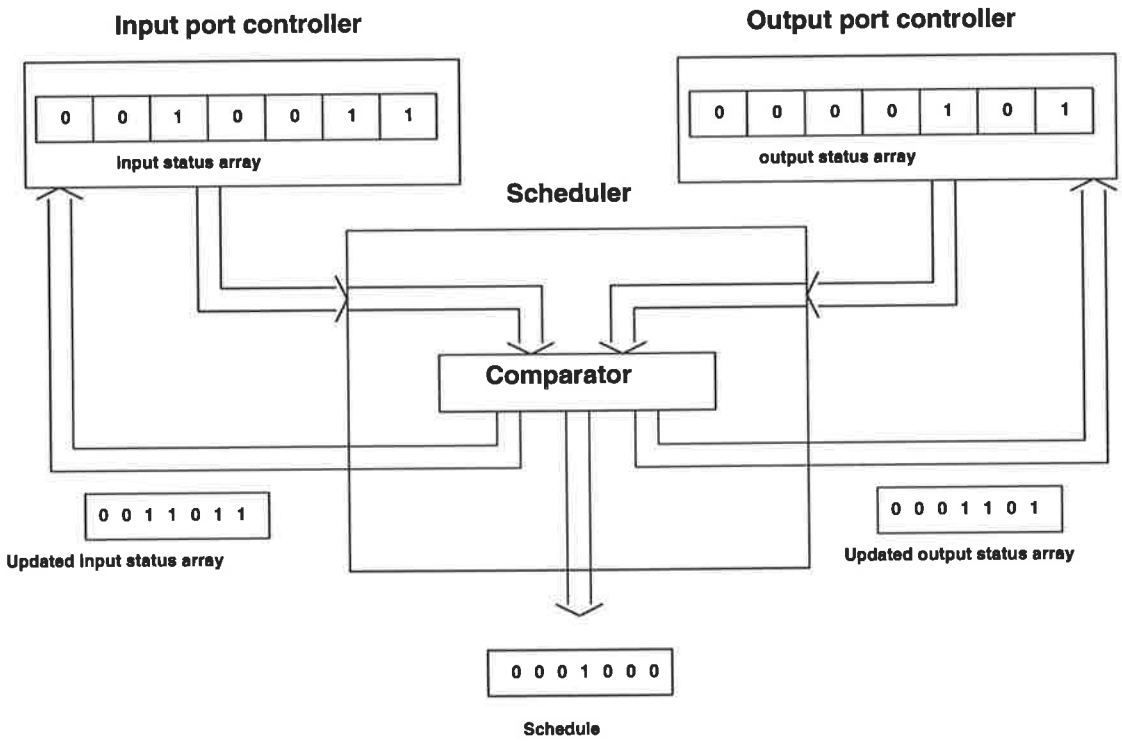

Figure 5 Diagram to illustrate the time scheduling algorithm

We can use a flow chart (Figure) to describe this algorithm.

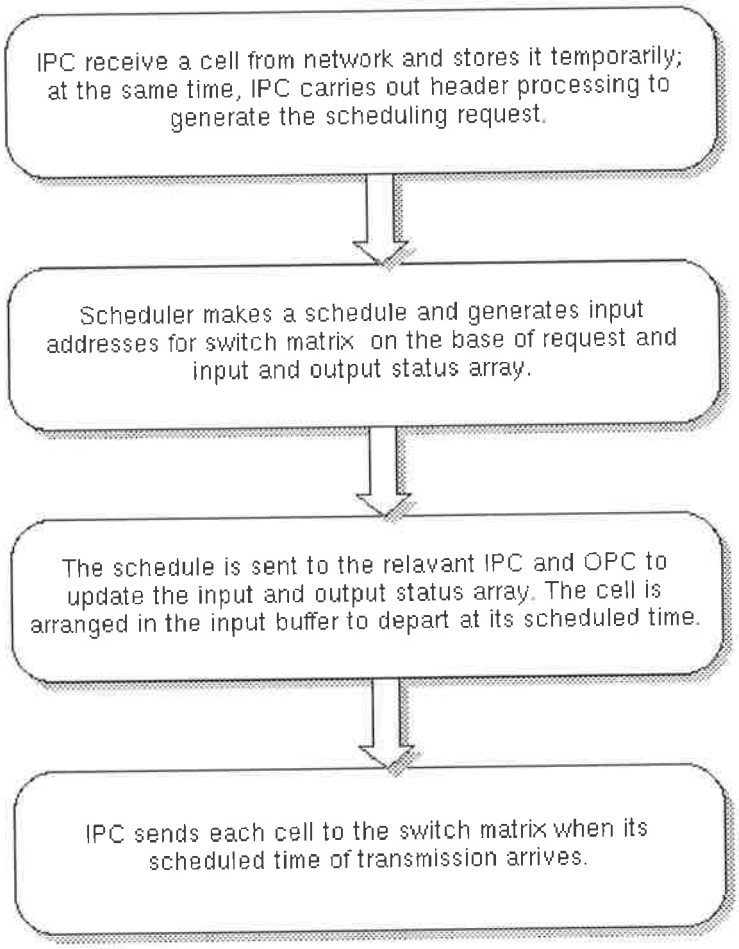

Figure 6 Flow chart of operation of switching process.

To make this a bit clearer, let us study such two arrays shown in the diagram. The input status array is "0010011" and the output status array is "0000101". We assume that the least-significant bits of these arrays represent the first time slot. When this input-output pair is requested by a cell, these two arrays are sent to the comparator in the scheduler. The comparator compares the two arrays starting from the first time

slot until a schedule is found. For the first bit, both of the arrays are *1* which means neither of them is available for a schedule. Then the scheduler compares the second bit, for this bit the input port has been allocated, so no schedule can be made. For the 3rd time slot, although the input port is available the output will be busy, so there is still no schedule. For the 4th one, both ports are available, so a schedule is made. Since the purpose of the comparator is to find the first time slot available for switching, the comparator stops as soon as a schedule is found.

The comparator generates a schedule result and sends the updated input status array to the input port controller. As show in the diagram, the schedule result is an array of binary numbers, in which the binary *1* represents the schedule. Also, the 4th bit of the input status array is updated to *1*. In addition, the comparator sends an updated output status array to the output port controller. In a similar way to the input status array, the 4th bit of the output status array is changed to *1* which means that this time slot has been scheduled out. This new output status array will be stored in the output port controller. After this time slot, both the input and output status arrays are shifted by one bit so that the second time slot becomes the first and so on. The bit corresponding to the 1st time slot is used to guide the input port controller to send the cell to the switch matrix. The bit corresponding to the 16th time slot is fed a *0* that means a new time slot is available.

This scheduling algorithm can effectively avoid conflict among cells that may otherwise use the same output port at the same time slot. This can decrease the loss rate and improve the delay performance significantly. Sarkies and Main [6] showed that this algorithm makes a throughput of greater than 90% achievable, when the

status length is sufficiently large. Therefore, this time scheduling algorithm can ensure high performance of the switch.

1.5.2 Enhancement

The algorithm can be enhanced by adding priority and multicasting.

1.5.2.1 Priority

Basically, there are two types of priority algorithm, namely delay priority and discard priority. For the first case, the cell with a low priority may suffer more delay, because the network always serves the high-priority cell first. Discard priority means that a cell with lower priority will be more likely to be discarded when compared with a high-priority cell. Usually, the discard priority is very simple and is easily implemented. The priority used by this algorithm is discard priority.

Specifically, for this algorithm a priority threshold is associated with each time slot. The high priority cell can use any time slot for scheduling, while the low priority cell can only be scheduled within the time slots that occur before the threshold. If a cell with a low priority can not make a schedule within the time slots permitted for it, the cell will be discarded.

Clearly, this approach is very simple to implement, but the penalty for it is less flexibility. In different ATM networks the threshold for the priority may need to be set to a different value. Therefore, to achieve the best flexibility and simplicity, the

scheduler should be designed to be able to support a variable threshold. In other words, the threshold value can be set to any point and this value is decided by the priority level.

1.5.2.2 Multicasting

The algorithm also supports multicasting. Multicasting means a cell from one input can be switched through a number of outputs simultaneous. In order to support multicast, the switch matrix should broadcast a cell to all the 16 outputs. Then input addresses are sent to their corresponding output ports, which select and connect one of the inputs to the output. Clearly, if a number of selector slices select the same input, the cell from this input will be multicast. We will discuss the crosspoint switch architecture that supports multicast in a later section.

1.6 Summary

In this chapter, we introduced the basic structure of an ATM switch. The switch matrix is the core of the ATM switching system. Therefore, we reviewed the crosspoint switch architecture. Then, we discussed particularly the structure of an input buffered ATM switch with a time scheduling algorithm. Finally, the time scheduling algorithm employed by this ATM switch is discussed.

In the next chapter, we take up these basic concepts and carry out the design of an ATM switch scheduler and its interfaces to the rest of the switch.

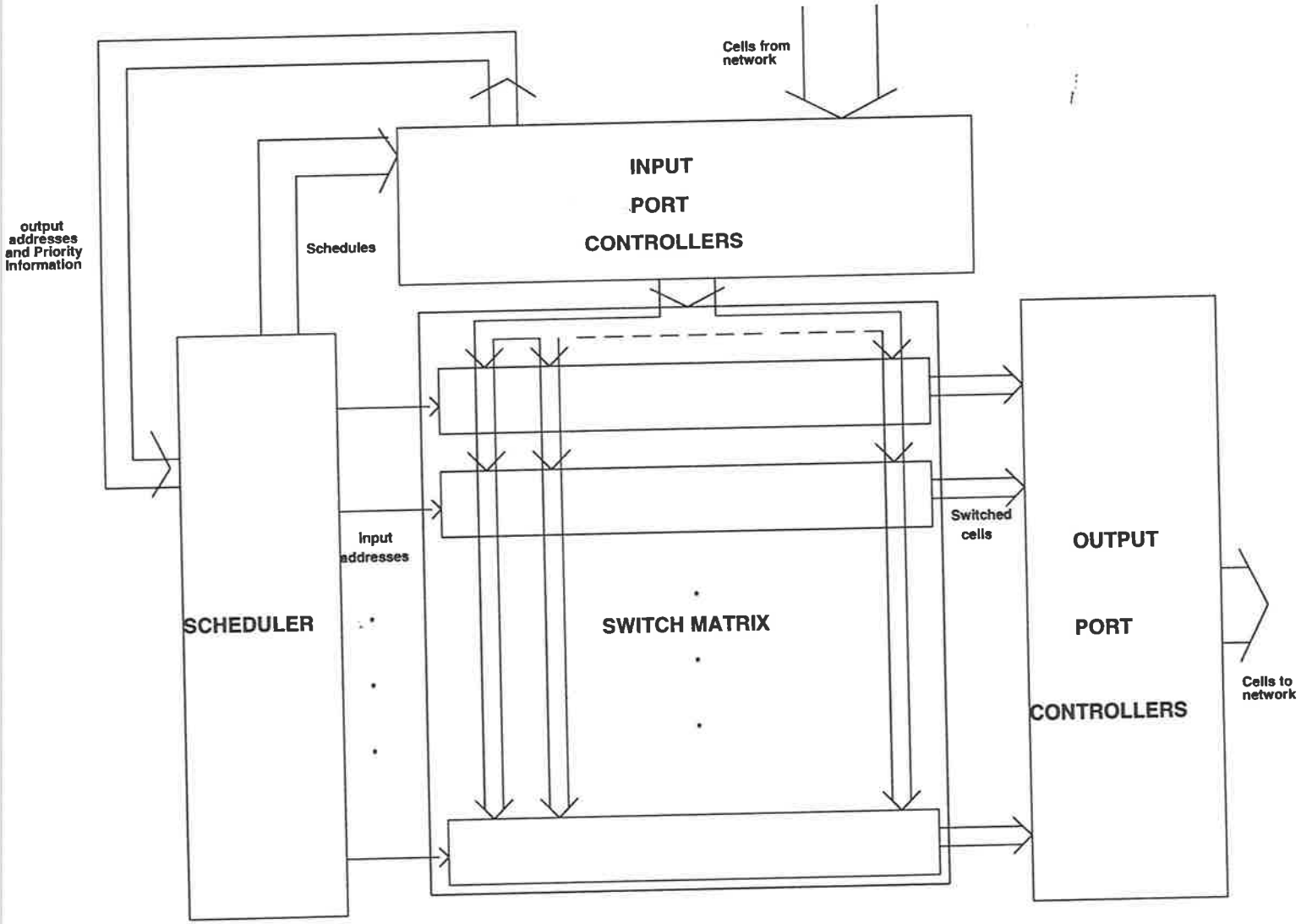

Figure 7 High-level architecture of an input buffered ATM switch

2.1.3 Interface Definition

The simulation results [6] of the time scheduling algorithm used in this switch show that the switch can achieve adequate throughput when 16 time slots are used for scheduling. Therefore, the scheduler discussed in this thesis is designed for scheduling within the 16 subsequent time slots.

Before we can discuss the detailed design of each functional block, we should be clear in what format each functional block interfaces with the others, because it will affect the structure of the basic functional blocks.

- **The Format of the Output Address**

As stated above, to make a schedule for a cell, the schedule needs a copy of the output address of this cell. In what form should we use to describe this output address? Recall that one of the design objectives of this ATM switch is to support multicast. Multicast means a cell from an input may be switched to a number of outputs simultaneously. Therefore, we need an array of binary number that can select any number of outputs. Towards that end, for an ATM switch with 16 output ports, we need a 16-bit binary number to describe the output address. In this binary array, we use *1* to represent a selected output port and *0* to represent a port not selected. In order to minimise the pin count for the scheduler, this 16-bit array is shifted into the scheduler in series. Therefore, the scheduler should have 16 inputs for the output addresses, each of the addresses coming from a different input port controller.

- **The Format of the Priority Information**

As we have discussed in the first chapter, for such a scheduler with 16 time slots for scheduling, we need a 16-bit array to describe the threshold value for priority. Note that we can also use a four-bit binary number to describe such a 16-bit array, and then the 4-bit binary number can be decoded into the 16-bit array. As will be shown in the later chapter, the scheduler should shift in and out the information at a very high speed, from a point of view of power saving it is advantageous to use the 4-bit addresses instead of 16-bit. Also, for this scheduler we assume that a cell to be multicast to the different outputs would have identical priority for all the outputs, because it would be very difficult to distinguish different priorities for different outputs of a multicast cell. Therefore, only one bit of priority information is required for any cell. Therefore, the scheduler has 16 inputs for priority information, and each one is connected to one input port controller.

- **The Format of the Input Address**

We have mentioned that the scheduler should generate the input addresses for each selector slice in the switch matrix to set up the data path. As with the output address, a question may be raised regarding the format of this input address. Since in one time slot one output port can only handle one cell, we should use such an array that can select one input from sixteen inputs. In order to minimise the hardware of the scheduler, we use a 16-bit array with one asserted bit to select an input. The input address is shifted out of the scheduler to the switch matrix in series.

- **The Format of the Schedule**

As we know, the scheduler should return a schedule for each input port controller as the result of request processing. This array of schedules informs the input port controller in which time slot the cell is to be scheduled. For the multicast case, a cell may be scheduled into a number of time slots to be switched out through different outputs. If this is the case, we have to use a 16-bit array to describe which time slots the cell is scheduled to. Therefore, the scheduler has 16 outputs for shifting the 16-bit schedule array back to the input port controllers.

2.2 Switch Matrix

As described above, the core of the ATM switch is a switch matrix, in which the data paths are developed between the input and output ports at the request of the cell. In the first chapter, we have reviewed some possible architectures. In this section, we will discuss the crosspoint switch architecture that meets the requirement of the particular scheduler discussed in this thesis. According to the above discussion, we note that the crosspoint switch suitable for this ATM switch should have the following characteristics:

- high speed;
- supporting multicast;
- interfaces perfectly with the other parts of the ATM switch.

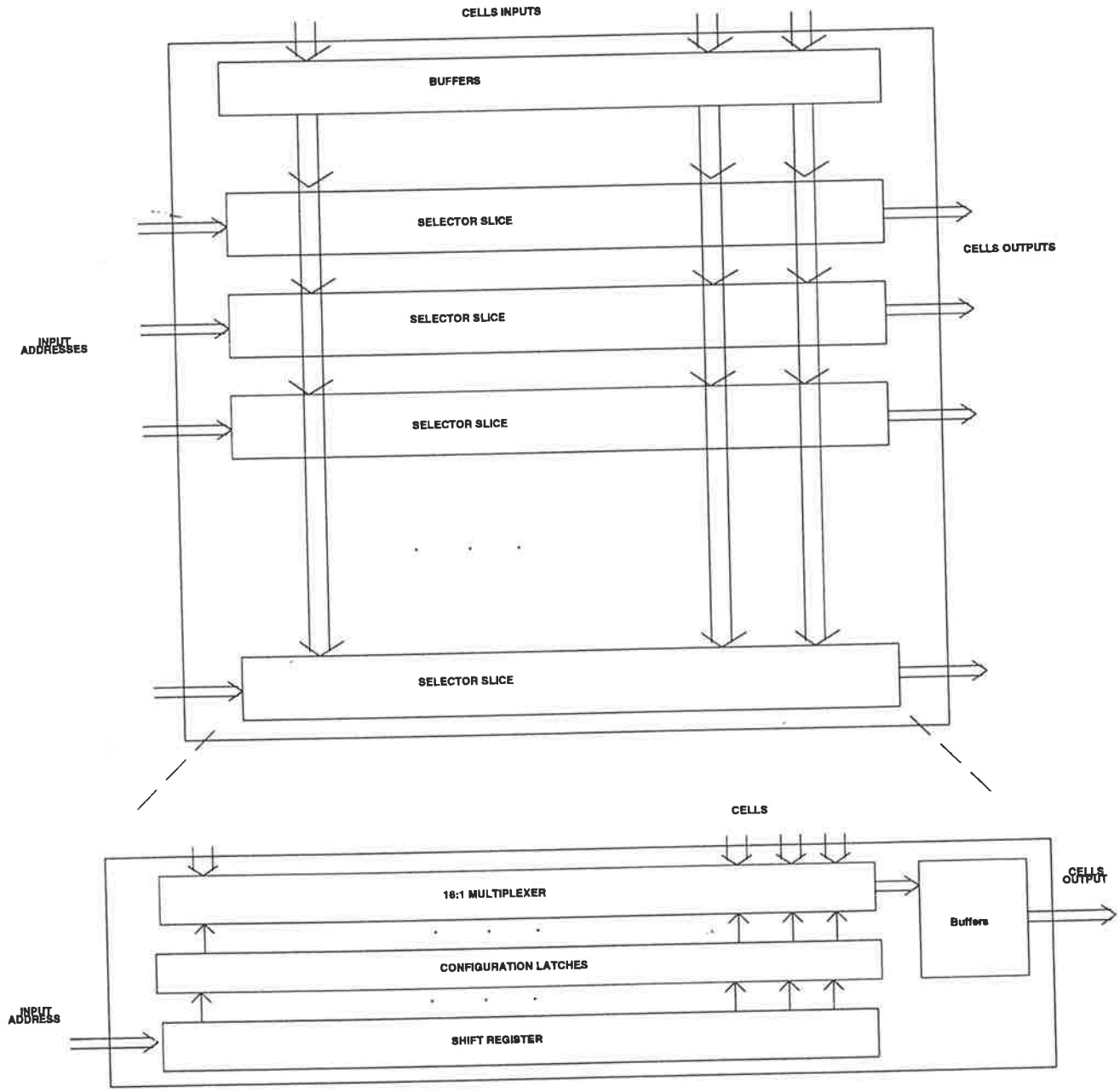

Figure 8 Architecture of crosspoint switch used in this ATM switch

Figure 8 shows a crosspoint switch architecture, which employs a broadcast and select architecture. This crosspoint switch consists of 16 cell inputs, 16 cell outputs and 16 address inputs. The switch is comprised of 16 selector slices, each of which corresponds to an output. The incoming cells are driven by the buffers and broadcasted to all the selector slices. Each selector slice receives an input address that informs the selector to select one of the cell inputs and connect it to the output.

The structure of the selector slice is shown at the bottom of the figure 5. The selector includes four parts: a 16-bit shift register, a configuration latch, a 16:1 multiplexer and a buffer. The input address is shifted into the shift register in series. When all the addresses are ready, the configuration latch is turned on, and the input address is loaded into the multiplexer in parallel. The asserted bit in the address will select one of the cell inputs and send it to the buffers. The buffers drive cells to the output port controller.

On the one hand, the broadcast and select architecture ensures that the switch can support multicasting and the simple structure makes very-high-speed switching possible. On the other hand, each selector slice receives a particular input address for its corresponding output port simultaneously, which means that the switch need only set up once per time slot. It simplifies the external control significantly. Moreover, we employ a 16-bit shift register to receive the 16-bit input address generated from the output scheduler, to ensure that the switch matrix can interface with output scheduler perfectly.

Currently, the GaAs crosspoint switch as described in the literature can offer the capability of up to 16 input and output ports at 10Gb/s for each [4], so the design of the scheduler and input port controller should be able to support such very-high-speed switch matrix.

2.3 Design of the Input Port Controller

In the ATM switching system, the input port controllers which act as the interface between the switch matrix and the output scheduler play a significant role in the data flow control. Now let us look at how an input port controller works. The input port controllers should coordinate with the output scheduler and switching fabrics to provide not only the basic cell control capability but also the advanced functions such as multicast and variable threshold for priorities.

2.3.1 Overview of the Input Port Controller

As we have discussed, the input port controller is the interface between the network and switch. On the one hand, the controller processes the cell header and deduces the necessary routing information for switching. On the other hand, the input port controller coordinates with the scheduler to manage the cell flow. To realise these two functions, the input port controller consists of two basic functional blocks: the header processor and the buffers. A diagram of the input port controller is shown in figure 9.

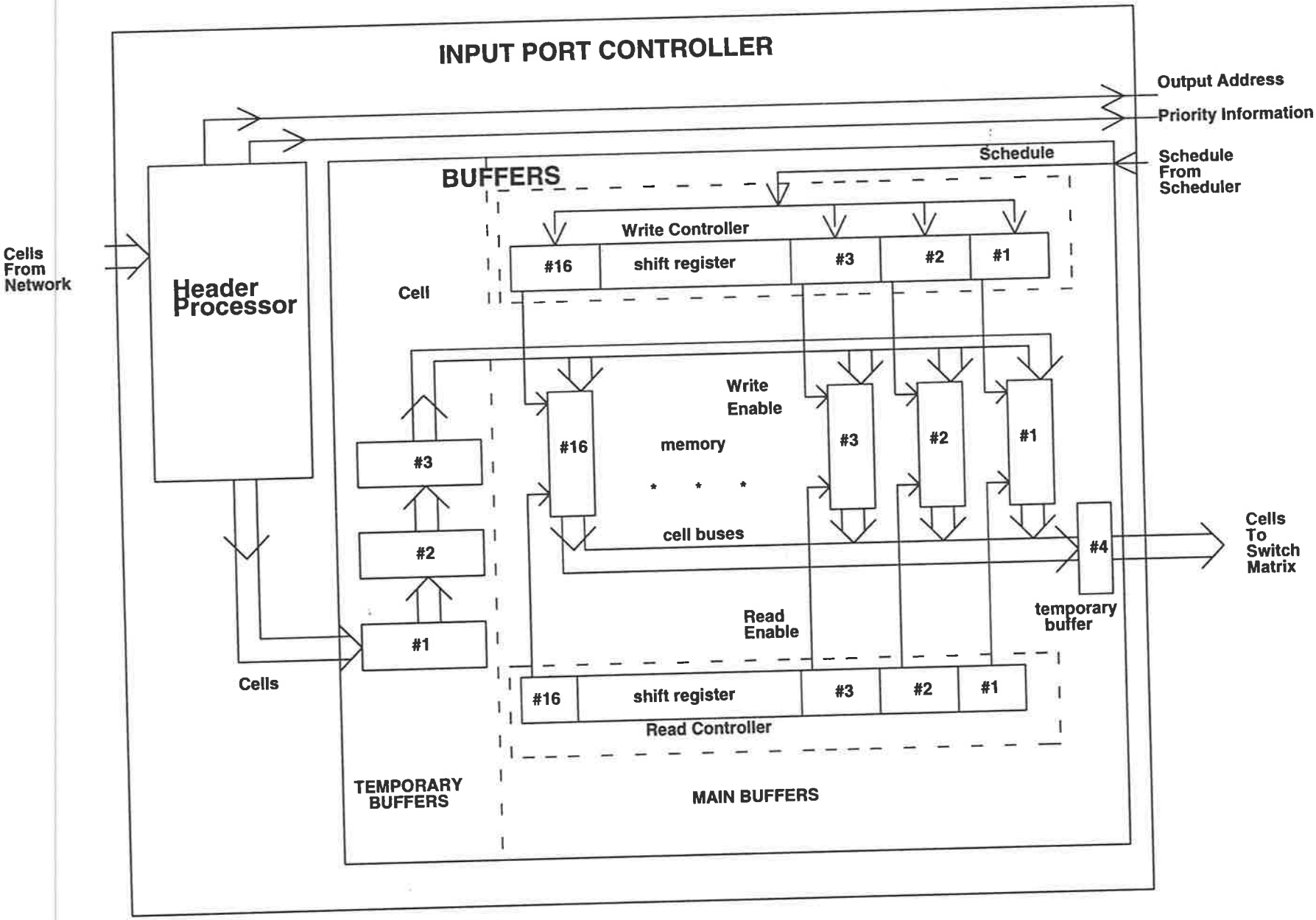

Figure 9 Diagram of input port controller

2.3.2 The Function of Header Processor

When a cell arrives at an input port controller, first of all, it goes through a header processor. Recall that in each ATM cell there is a header that consists of the properties and the routing information of the cell. The header processor analyses the header of the cell so as to determine the type of the cell. If the idle cell that is inserted by the physical layer and contains no user information is detected, this cell is discarded immediately. However, for the user cell the header processor generates the necessary information for switching the cell. The routing information is updated and the cell is passed to the buffers.

As we have discussed in the previous section, for this ATM switch the information required to switch a cell is the output address that indicates to which outputs the cell is to be switched, and the priority information that describes how many scheduling resources can be used by the cell. Recall that because of the need of multicast, the output address should be in the form of an array of 16-bit binary numbers. Each bit of this array corresponds to an output and an asserted bit represents an output for the cell to be switched out. The priority information should be a 4-bit binary number, which will be decoded to a 16-bit array in the scheduler. The header processor sends them to the output scheduler respectively.

2.3.3 The Operation of the Buffers

The buffers in the input port controller not only store the cells but also control the cell flow. The diagram of the buffers is also shown in figure 9. From the diagram we

stored in the #1 buffer, the cell stored in the #1 buffer is forwarded into the #2 buffer. Similarly, at the beginning of the 3rd time slot, the cells in the #1 and #2 buffers are forwarded into the #2 and #3 buffers respectively and a new incoming cell is stored in the #1 buffer. At the end of the 3rd time slot, the schedule is received, and the cell in the #3 buffer is written into the main buffer under the guidance of the schedule result.

2.3.3.2 The Structure of the Main Buffers

The block diagram of the main buffers is shown in figure 9. The main buffer consists of two basic blocks: one is the memory, the other is the read-write controller.

The memory is comprised of 16 memory units. Each unit can store a cell. The capability of the main buffer is determined by the scheduling capability of the output scheduler. Recall that this output scheduler is designed to be able to make a schedule within the subsequent 16 time slots. As we will see later, storing a cell into the main buffer is dependent on the schedule for it. If no schedule is made for a cell, this cell will not be written into the main buffer at all. The cell that fails to enter the main buffer will be discarded. Therefore, the number of the memory units in the main buffer is identical to the maximum number of the cells that can be scheduled. In addition to 16 memory units, there is a temporary buffer in the main buffers, which receives the cell from the cell buses and sends it to the switch matrix after one time slot. The necessity of this temporary buffer will become evident shortly.

Associated with the memory, there are two shift registers, which are the write controller and read controller respectively (see figure 9). Each bit of the shift register is connected to a memory unit so as to control the write and read process.

2.3.3.3 Operation of the Main buffer

In the last subsection, we mentioned that the operation of the main buffer relies on the schedule results. Therefore, we can associate the position of the memory unit with the time slot. Specifically, each memory unit corresponds to a time slot. Thus, we can identify the memory address with the time slot.

We shall now discuss the operation of the main buffer. Similar to other memory devices, it includes two processes: read and write.

- **Reading Process**

As we know, the cells are scheduled into the subsequent 16 time slots. When a cell is sent out, the subsequent cell becomes the leading cell in the new time slot. Therefore, when a cell is saved in a memory unit, the time slot to which this memory unit corresponds will change with the time. For example, at one moment, the #1 memory corresponds to the first time slot. The #2 memory corresponds to the second time slot (refer to figure 9). In the next time slot, the time slot that the #1 memory corresponds to becomes the current time slot and the cells in #1 memory is sent out. At the same time, the #2 memory corresponds to the 1st time slot and all the other groups now correspond to its subsequent time slot. After this time slot, the cell in the

#2 memory will be sent out and #3 memory will correspond to the 1st time slot. Therefore, we need a pointer that can indicate which memory corresponds to the current time slot and activates it to send the cell out.

In order to select the cell to be sent out, we need a read controller. Indeed, this controller is a circular 16-bit shift register with the input of the first bit connected to output of the last bit (see figure 9). Thus, the state of this shift register can be rotated around the register. Each bit of this 16-bit shift register is connected to the "read enable" input of a memory.

When the controller is initialised with an array of states such as "0000_0000_0000_0001". The shift register is designed to shift the state in it in a counterclockwise direction, namely, the state in each bit is moved to the bit on its left and the left-most bit is shifted into the right-most bit. The shift register is driven by a signal that is asserted at the beginning of each time slot. Clearly, the state *1* will be moved around one bit per time slot. We can regard the state *1* in the shift register as a pointer.

As stated above, each memory unit corresponds to one of 16 subsequent time slots, so the memory unit that has just sent the cells out should correspond to the 16th time slot, a new time slot for scheduling. Therefore, we conclude that the pointer is always pointing to the memory unit that corresponds to the 16th time slot.

Since the pointer is always moved counterclockwise, the memory unit on the left of the pointed one should correspond to the 1st time slot. These properties are very important for the operation of write controller. We will come back to this point later.

If we have been clear about the properties of this pointer, it is very easy to understand the operation of the reading process. The pointer is moved around the register one bit per time slot. The memory pointed by a pointer will send the cell to temporary buffer #4 the cell buses (refer to figure 9). After one time slot, the cell in buffer #4 is forwarded into the switch matrix. Then a read process is finished. We will explain the reason of using this buffer #4 in a later section.

- **Writing process**

The write process is to save the cell held in the #3 temporary buffer into the main buffers. The memory in which the cell is written is controlled by the write controller. As shown in figure 9, the write controller is also a 16-bit shift register with the input of the first bit connected to the output of the last bit. The shift register receives the schedule in series and sends them out in parallel. Each bit of this shift register is connected to the "write enable" input of a memory unit. This shift register is somewhat different from a common series-in-parallel-out register. We note that the schedule input is directed to all the 16 bits of the register. That means any bit of the register can be selected as the entry-point for shifting in the schedule array, but only one is selected at any one moment. The entry-point of the register is decided by the pointer.

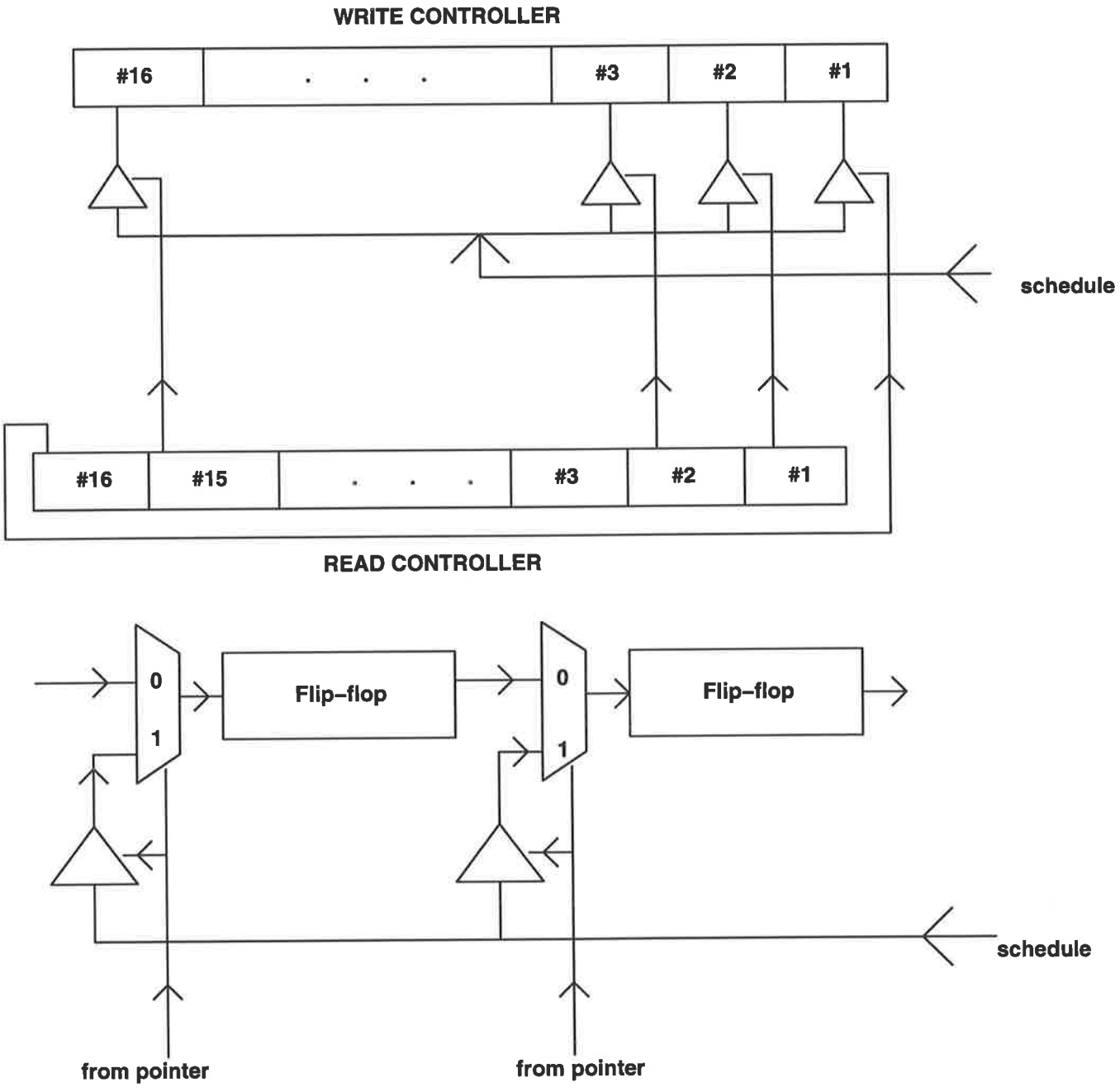

Figure 10 Structure of write controller

Part of this shift register is shown at the bottom of figure 10. Note that there is a 2:1 multiplexer between each flip-flop. The control signal of this multiplexer comes from the pointer. When the pointer is *1*, the input of the flip-flop is connected to the schedule, so the flip-flop pointed by the pointer becomes the entry-bit and the schedule shifts into the shift register. At the same time, the multiplexer blocks the signal from the preceding flip-flop. On the other hand, those flip-flops that are not pointed to by the pointer will receive the signal from its preceding stage, whose operation is identical to an ordinary shift register. The reason for using such a structure will become evident shortly.

Note that the output of the #3 temporary buffer is connected to the input of all the 16 memory units in the main buffer. That means the cell can be sent into any number of enabled memory units simultaneously. This is due to the need for multicast, in which the cell from one input may be scheduled to a number of different time slots. Since we associate the memory place with the time slot, it makes a good sense to save the cell into the memory unit that corresponds to the time slots scheduled by the cell. For example, if a multicast cell is scheduled into both the 1^{st} and 2^{nd} time slots, we should save the cell into the group that currently corresponds to the 1^{st} and 2^{nd} time slot. Clearly, due to this association relationship, we can use the schedule array to select the memory units and write the cell into them. Specifically, the asserted bits in the schedule array represent schedules. The asserted bits will turn on the "write enable" of certain memory units and the cell is written in.

If we use the schedule array as the write address of the main buffers, we have to make sure that each memory unit and its received schedule correspond to the same

time slot. The time slot to which each memory unit corresponds is always changing, which is under control of the pointer. Therefore, a write controller is necessary to select the entry-point of the schedule.

The write controller is a shift register with variable shift-in bit. Here we assume that the schedule array is always shifted into the register counterclockwise. For example, the schedule result in #2 bit register is shifted to #3 bit; the schedule result in #3 bit is shifted to #4 bit; the schedule result in #16 bit is shifted to the #1 bit. In addition, we assume that the most-significant bit of the schedule array is shifted into the register first, and the least-significant bit is shifted in last.

On the basis of the above assumption, we note that no matter from which bit the schedule array is shifted in, the final result will follow such regularity: the schedule result that corresponds to the first time slot (least-significant bit) is always stored in the entry-bit register; the schedule result that corresponds to the 2^{nd} time slot is stored in the register to the left of the entry-bit; the schedule result corresponding to the 16^{th} time slot (most-significant bit) is stored in the register to the right of the entry-bit. An example is shown in figure 10. The diagram shows the entry-point, shifting direction and the final result. For this example the entry-point is the #3 register. When all the bits are shifted in, the bit corresponding to the first time slot is stored in the #3 register. The bit corresponding to the 2^{nd} time slot is in the left of the entry-point, #4 register, and the bit corresponding to the 16^{th} time slot is in the right of the entry bit, #2 register.

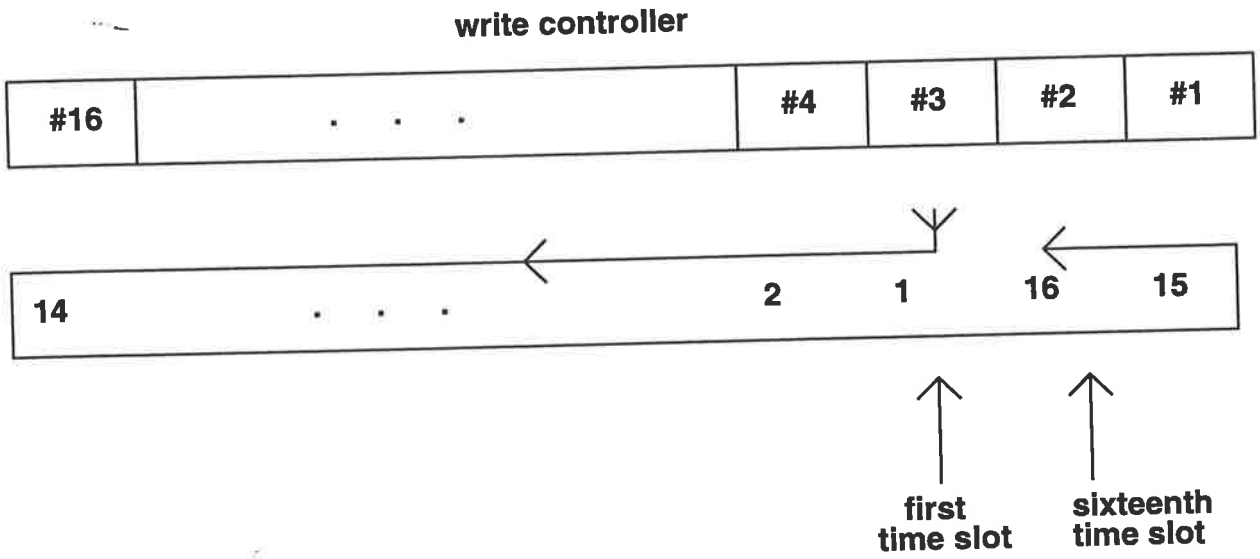

Figure 11 Operation of the write controller

We have concluded that the time slot to which a memory unit corresponds is decided by the pointer in the read controller. The memory unit referenced by the pointer always corresponds to the 16^{th} time slot and its left one corresponds to the first time slot. Therefore, we can simply use the read controller to select the entry-point of the schedule in the write controller. A diagram is shown on the top of figure 10. We note that each output of the read control is connected to a tristate buffer. The tristate buffer that is turned on will develop a path for the schedule to the write controller and the register connected to this buffer will become an entry-point. We note that the #N bit of the read controller is connected to the #(N+1) bit of the write controller. In other words, when the pointer of the read controller is pointing to the #N bit, the #(N+1) bit in the write controller will be selected as the entry-point for the schedule.

The writing operation is very straightforward. The pointer in the read controller turns on a tristate buffer. The schedule is shifted into the write controller from the selected entry-point. At the end of the time slot, all the schedules are ready in the write controller and they are loaded into the memories. The cell from the temporary #3 buffer is written into the selected memory units. The write process is accomplished.

2.4 The Design of the Output Scheduler

As discussed in the first chapter, the most advantageous features of this ATM switch are due to the time scheduling algorithm, and this algorithm is realised by the output scheduler.

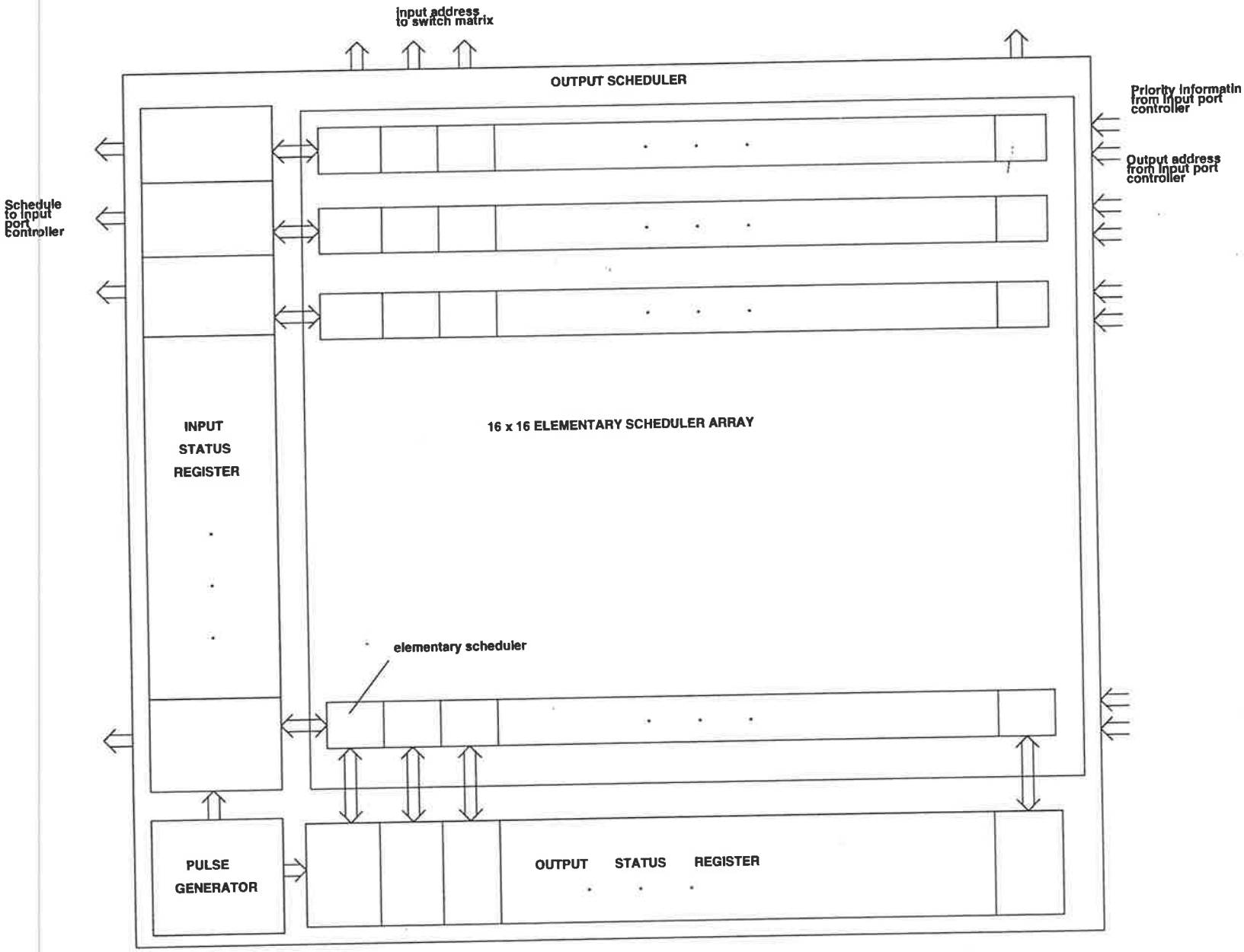

Figure 12 Structure of output scheduler

2.4.1 The Structure of the Output Scheduler

A functional block diagram of the output scheduler is shown in figure 12. The example output scheduler shown in the diagram, has 32 input ports and 32 output ports. The inputs include 16 output addresses and 16 priority information codes. All of this input information comes from the input port controller. There are 16 output ports connected to the 16 input port controllers, which send the schedules to the controllers. The other 16 output ports are connected to the switch matrix. Each of these ports passes to the switch outputs, the input address to which the switch output is to be connected.

Basically, this output scheduler is composed of four main functional blocks: the elementary scheduler, the input status register, the output status register and the clock generator. As shown in the diagram, the scheduler consists of a 16x16 array of elementary schedulers. Each row of the elementary schedulers corresponds to an input, and each column of the elementary schedulers corresponds to an output. Each elementary scheduler corresponds to an input-output pair. The function of the elementary scheduler is to compare the input status array with the output status array.

To minimise the amount of information to be exchanged between the output scheduler and outside, we can simply maintain the input status array and output status array within the output scheduler. The input status array and output status arrays are stored in the input status register and output status register respectively (see figure 12). In addition to storing and updating the input status array, the input status register is in charge of sending the schedules out to the input port controller.

Moreover, a 4:16 decoder and a 16-bit address register are also integrated into the input status register, although they have a separate function. Another part of this output scheduler is a clock generator. This generator receives the clock signal from outside and it buffers and distributes it every clocked circuit on the chip. It also generates the assertion pulses which are needed both by the status register and the elementary schedulers.

2.4.2 Operation of the Output Scheduler

A more detailed diagram of this output scheduler is given in figure 13, which shows dataflow among each functional block. The aim of this section is to illustrate how each functional block interfaces and coordinates with others. The detailed design within each functional block will be discussed in the next section.

2.4.2.1 Data flow of the Output Scheduler

In this section we will discuss how the output scheduler works. Before we step into the detailed discussion of operation, a flow chart (see figure 14) can help us to understand the basic function of scheduler.

The operation of an output scheduler can be divided into three steps and each step takes one time slot. The first step is to import the request from the input status register; the second step is to compare the status arrays; the last step is to send the scheduling result out. Clearly, in order to achieve the best efficiency, shifting in the

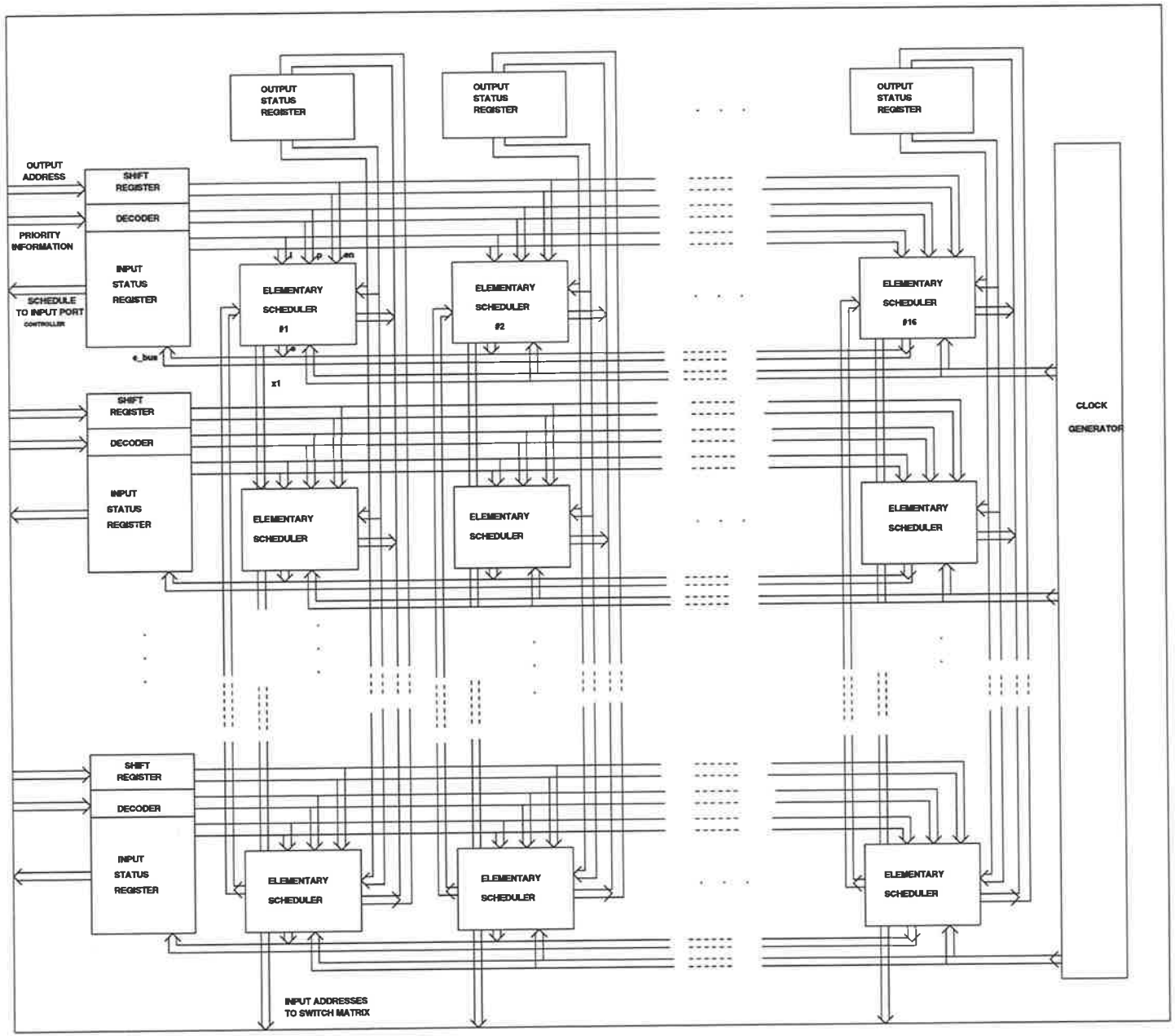

Figure 13 Detailed structure of output scheduler

output addresses and priority information, comparing the status arrays and sending the input addresses and schedules out are all carried out in a pipelined manner.

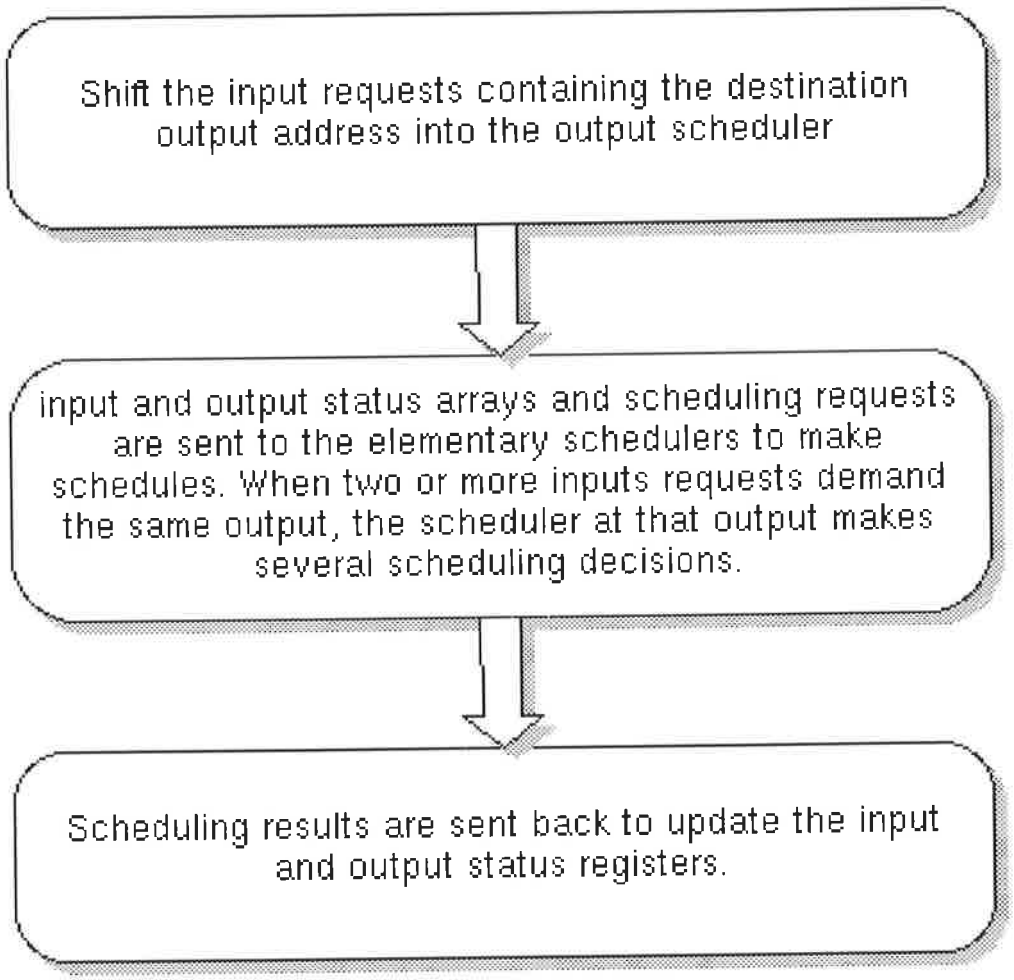

Figure14 Flow chart of operation of output scheduler

- **Importing the Request**

As mentioned above, the function of the output scheduler is to compare the input and output status arrays so as to find the first time slot available for both the requested input and output. The status arrays are compared within the elementary scheduler that corresponds to the requested input-output pair, so we need some information to find the particular elementary scheduler. Moreover, any schedule is made on the basis of the priority information, so some circuits must be provided in the output scheduler to handle this priority information.

Recall that each row of the elementary schedulers corresponds to an input and each elementary scheduler within this row corresponds to an output. Therefore, we can use an output address to select particular elementary schedulers and activate them to compare the status arrays. At the beginning of this chapter, we have stated that in order to support multicast (in which an input can select a number of outputs simultaneously), a 16-bit array is used to describe the output address. Referring to figure 13, in each row of the elementary schedulers, there is a 16-bit shift register that receives the output address from the input port controller in series and sends it to each elementary scheduler in this row in parallel. Each bit of this shift register is connected to the "enable" input of an elementary scheduler. The asserted bit in the address will activate the corresponding elementary scheduler to perform the comparison of the status array.

Let us study an example. If a cell requests to be switched to the outputs labelled #1 and #16 (see figure 13), its output address will be an array of bits such as

"1000_0000_0000_0001". It takes one time slot to shift it into the shift register. At the beginning of the next time slot, the address is sent out in parallel. Each bit of this array is sent to the corresponding elementary scheduler. In this example, the elementary schedulers corresponding to the first and last output receive a logical *1*, so they are turned on and will compare the input and output status array. In contrast, the other elementary schedulers that receive *0* are turned off. In other words, when the status arrays flow through them they perform no operation and keep the status arrays intact.

Referring to figure 13, we note that we have provided a decoder with each row of the elementary schedulers. This decoder is utilised here to turn the 4-bit binary number that describes the priority information into a 16-bit array. As defined at the beginning of the chapter, we use a 4-bit array to describe the priority information, which is shifted into the output scheduler in series. This priority information describes the threshold value of the time slot that a cell can use. As shown in a later section, the elementary scheduler needs a 16-bit array to be used conveniently. Hence, we should turn this 4-bit binary number into a 16-bit array. This is done by the 4:16 decoder.

As we have discussed, in this output scheduler we assume that a cell to be multicast should possess the identical priority information. Therefore, we can simply send the decoded priority information for a multicast cell to all the 16 elementary schedulers in a row (see figure 13).

- **Comparing Status Arrays**

When both the output addresses and priority information is available, the output scheduler is ready for the next step which is request processing. This step is the essential step for the operation of the output scheduler. We discuss its basic operation first, then we will analyse its drawbacks and improve it.

At the beginning of the next time slot, both the output address and the decoded priority information are sent to the elementary schedulers. Simultaneously, the input status registers load the status arrays into all the elementary schedulers in its corresponding row. The output status registers send the output status arrays to one of elementary schedulers in its column (refer to figure 13). The output status array will go through all the elementary scheduler in this column one by one and the output status array is compared with the input status array in the activated schedulers.

Note that the operation of each column of the elementary schedulers is identical, so let us study the operation of one column, which is sufficient to mirror the operation of the whole output scheduler. For simplicity, we assume that the output status array is loaded into the elementary scheduler in the first row. We will have a more general discussion on this topic in later section.

For example, if the first elementary scheduler in a column is activated, the output and input status is compared in it. If there is a schedule made, the elementary scheduler would update the output status array and send an updated copy of the output status array to its subsequent elementary scheduler, the one in the second row. In figure 13 the updated output status array is labelled *X1*. Simultaneously, the

elementary scheduler generates a schedule array (labelled *e* in the diagram) in which *1* represents a schedule and *0* means no schedule. This schedule array will coordinate with other schedule arrays generated by the other elementary schedulers in this row to produce a final schedule array for the input status register.

The updated output status array *X1* is sent to the elementary scheduler in the second row. If this elementary scheduler is also turned on, the output status array will be compared with the input status array that is stored there. Assume that in this comparison no schedule is made, so the output status array will be kept intact and go to the next elementary scheduler. At the same time, the elementary scheduler outputs a schedule array. Since no schedule is made, it is just an array of zeros.

Subsequently, assume that no other elementary schedulers in this column are turned on, then the output status array will flow through each of them and return to the output status register with a value that is identical to the value of *X1*. Since no schedule can be made in a disabled elementary scheduler, the schedule outputs of elementary schedulers are all arrays of zeros.

At the end of this time slot, the input status registers fetch the value from the *e_bus*. Indeed, the value on the *e_bus* is the logical OR of all the schedule results corresponding to the same time slot and different output. We will explain later why a logical OR should be used. The input status register uses the scheduling results to modify the input status arrays stored in them. On the other hand, the output status register stores the new output status array in it.

- **Exporting the Scheduling Results**

As we know, the input port controller needs the schedule array to place the cell into the buffer and the switch matrix needs the input address to switch the cell. We have discussed the generation of the schedule result. We will discuss the generation of the input address in a later section. The third step is to send them out. At the end of the 3rd time slot, the output scheduler processes a request.

2.4.2.2 Structure Analysis

In the last section, we discussed how the output scheduler works. In this section, to gain insight into the operation of the switch, we will discuss why it is appropriate for the schedule to operate in this way..

One question may be why the input status array is sent to all the elementary schedulers in a row, while the output status arrays are fed to one scheduler and it ripples through the whole column.

As the name "output scheduler" implies, all the schedules made in this scheduler correspond to the output ports. In other words, what is scheduled is the operation of the output port. One output port can only handle one cell at one time slot, so it demands the output scheduler to schedule only one cell into a time slot. In the output scheduler, the output status arrays are employed to indicate the state of the output ports. If there were a schedule made the output status would be updated to *1*

immediately. The *1* in the output status array represents a time slot that has been scheduled out. Therefore, this *1* will give its subsequent elementary scheduler no chance to make further schedules into that time slot. Thus, this structure ensures that only one cell is scheduled to a time slot, in other words, no conflict may occur on the output port.

On the other hand, consider the input status arrays. As we know, this switch should provide the multicast function, so it is not surprising for one input port to send a cell to a few output ports simultaneously. In the section discussing the operation of the switch matrix, we have seen that in the crosspoint switch the inputs just broadcast the cells to all the outputs, and the output will select one of them according to the input address that comes from the output scheduler. This architecture implies that from the point of view of the input port of the switch matrix there is no difference between switching the cell to one output or to sixteen outputs. Therefore, we just send the input status array to all the elementary schedulers in a row simultaneously. Each elementary scheduler in this row will generate an array of schedule results. Therefore, we use a logical OR of all the corresponding bits of the 16 arrays as the final schedule result and send it to the input status register. This explains why the final schedule result should be the logical OR of all the individual results.

2.4.2.3 Defeating Unfairness

Now we have understood the necessity of rippling the output status array through the column of elementary schedulers, another question related to it will appear: since each elementary scheduler can only receive the updated output status array from its

upper one, the upper elementary scheduler has a greater chance to make schedules as it receives the output array earlier. In other words, there is unfairness between each elementary scheduler in a column. In order to solve this problem, we have to change the entry-scheduler of the output status array frequently and regularly so that each elementary scheduler in a column has an equal opportunity on average to achieve the best service as well as the worst service.

Towards that end, the scheduler is designed to be able to load the status array into any one of the elementary schedulers in one column and consequently, the updated output status array can be returned from any elementary scheduler (see figure 13). Clearly, in one time slot only one of them should be turned on to receive and return the status array. Therefore, we need a signal to select it. We employ a 16-bit shift register to act as a pointer in the signal generator. We have discussed the operation of a pointer in the previous section. Each bit of this register is connected to a row of elementary scheduler. When the pointer is pointing to a row, the row below the pointed row would be the entry scheduler of the output status arrays for this time slot. Thus, it receives the output status array from the output status register directly. The other ones will receive the status array from its upper scheduler. When the output status array goes through all the 16 elementary schedulers in a column, it is returned to the output status register through the scheduler in the row pointed to. The entry elementary scheduler is changed in each time slot, so that after 16 time slots, each elementary scheduler in a column achieves the identical average chance for service.

2.5 Summary

In this chapter, we discussed the operation of the ATM switch at the highest level. Four basic functional blocks are introduced and their coordination with each other is demonstrated. Then, we discussed the structure and operation of the input port controller. From the discussion of the input port controller, we understand how the controller generates the request and manages the cell flow according to the schedule results. Subsequently, we discussed the architectural design of the output scheduler, from which we understand the basic scheduling process. With a good understanding of the high-level operation, we will now investigate the detailed design in a lower level.

Chapter Three

Investigating Basic Functional Blocks of the Output Scheduler

Now that we have discussed the architecture of this output scheduler and studied how each module coordinates with others, we are ready to have a close look at the operation of each functional block.

3.1 Elementary Scheduler

3.1.1 Structure

First of all, let us study the interface of an elementary scheduler with the outside world. The block diagram of an elementary scheduler is shown in figure 15. Since the basic function of an elementary scheduler is to compare the input status array and output status array, each elementary scheduler has two 16-bit inputs for the input and output status array respectively. As we have mentioned, the priority information also takes part in the comparison, so a 16-bit input for priority information that is decoded from a 4-bit binary number is needed. The elementary scheduler also needs an "enable" input which is controlled by the output address. In addition, in order to defeat unfairness, each elementary scheduler is designed to be able to receive the output status array directly, so a 16-bit input for the output status array and a pointer

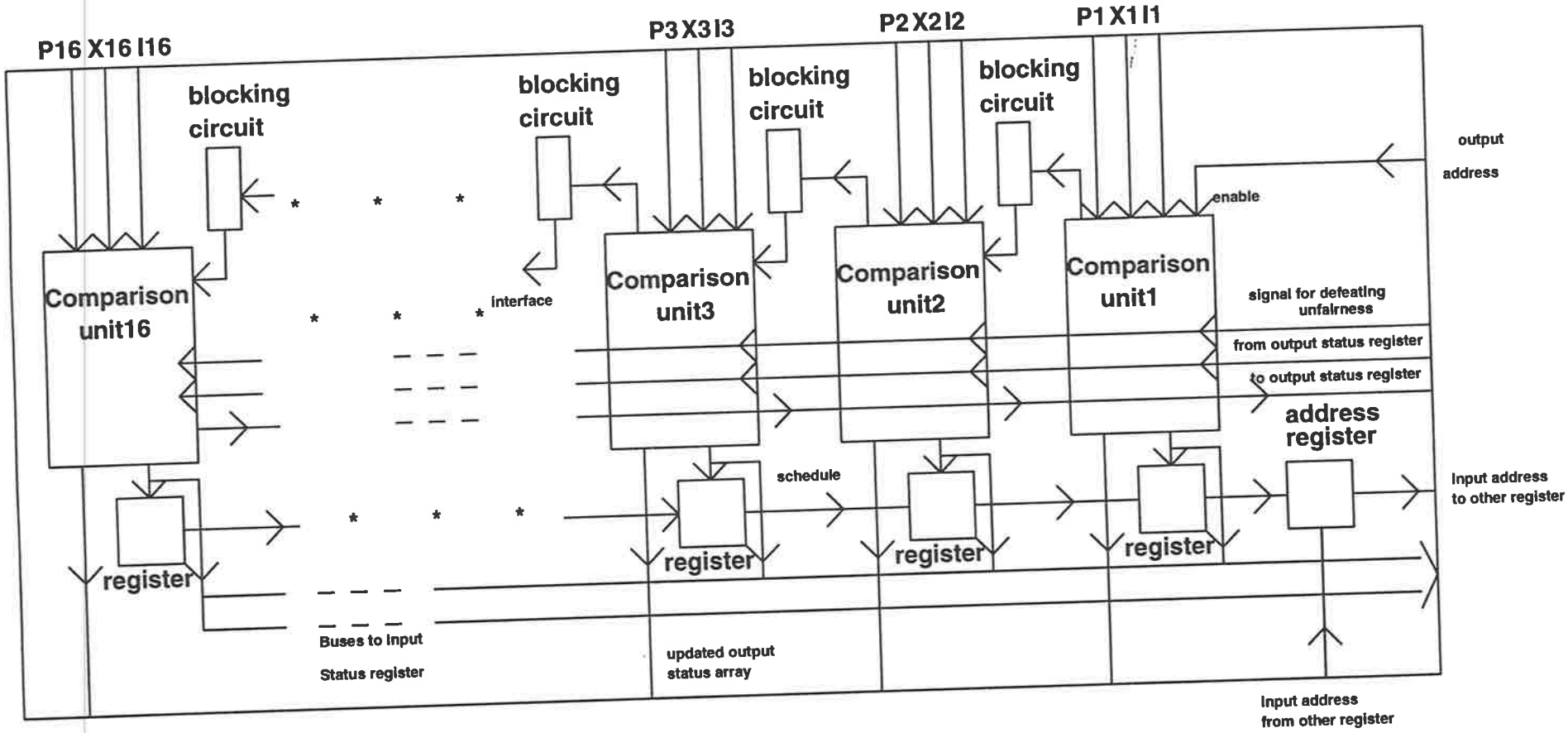

Figure 15 Diagram of the elementary scheduler

input are necessary. Each elementary scheduler has a schedule array output and two output status array outputs one of which is connected to the subsequent elementary scheduler and the other is connected to the output status register. Moreover, each elementary scheduler has a one-bit input and a one-bit output for the input address.

Secondly, let us consider the internal structure of an elementary scheduler. The elementary scheduler consists of two main parts: the comparators and the schedule registers. As shown in figure 15, there are 16 comparison units and each unit corresponds to one time slot. For each unit, there are four inputs: input status, output status, priority information and the interfacing signal. Note that associated with each comparison unit there is a blocking circuit, which generates the interface signal to its subsequent unit. As we know, the scheduler is only concerned with the first time slot in which both input and output port are available. The blocking circuit is employed to keep other schedules from being reached once the first one is found. For the first unit, the interface signal comes from the output address, which determines whether to enable this elementary scheduler. In addition, each scheduler outputs a schedule array and an output status array. The schedule result has two branches of outputs, as shown in figure 15. One of them is sent back to the input status register. The other is fed into a 16-bit schedule register, whose function will be discussed later. Moreover, as shown in the diagram, there is a separate one-bit register, which is used for the generation of input addresses.

3.1.2 Operation of the Comparison Units

Whether an elementary scheduler will operate or not is decided by the "enable" signal, which is connected to one bit of the output address. If the output to which the elementary scheduler corresponds is not requested, a logical *0* will be sent to it. Then this signal is rippled down to each comparison unit through the blocking circuit and disables them. Alternatively, a logical *1* will activate this elementary scheduler. At the beginning of each time slot, the input status array and the priority information are sent to each elementary scheduler, while the output status register will not be available until it ripples to the elementary scheduler. When all these three sets of inputs appear on an elementary scheduler, the scheduler is activated. What we are interested in is the first time slot that is available for both input and output, so the comparison operation begins from the unit corresponding to the first time slot and passes along to the one corresponding to the last time slot. In other words, no decision can be reached for a comparison unit until it receives the interfacing signal from its preceding unit.

Now let us study a simple scheduling example without consideration of priority information. Assuming that an input status array, 0000_0000_0111_1101, and an output status array 1111_0000_0000_1111 appears on the input of an activated elementary scheduler. Here we regard the least significant bit as the first time slot. The first bit of each array is *1* which means that in this time slot both input and output are busy. Then the first unit's blocking circuit sends a logical *0* to the following unit to inform it that no schedule is yet found. With this signal, the second comparison unit is activated and begins to compare the second bit of the arrays. However, there is still no schedule found, so similar to the first one it passes the

responsibility to its next unit. Thus, the comparison is carried out one by one until a schedule is found. For these two arrays, in the 8^{th} time slot both arrays are *0* that means both input and output are available. The 8^{th} comparison unit generates a logical *1* as the schedule result and updates the 8^{th} bit of the output status array into *1*. Simultaneously, its blocking circuit sends a *1* to its subsequent unit and this signal will be passed through all its subsequent units. This signal prevents the subsequent units from making any schedules, although both ports are available in the 9^{th} to 12^{th} time slots.

3.1.3 Modifications for Priority

From the last example we are clear about the basic operation of the comparator. Now let us look at how the priority information affects scheduling. In the last example, we did not consider the effect from the priority information. We can regard this situation as that each cell has high priority, in which case the cell can be scheduled into any one of the 16 time slots. In this case, the priority information should be "1000_0000_0000_0000". As assumed above, the most significant bit corresponds to the 16^{th} time slot, so this priority information means the cell can not use the time slots after the 16^{th}. Thus all the 16 time slots are available for it.

A cell with a low priority can only use some of the time slots, which is decided by the threshold value set by the priority information. Let us still consider the two arrays in the last example to study how the priority information affects the scheduling results. For convenience, we write it here again, the input status array: 0000_0000_0111_1101, and output status array: 1111_0000_0000_1111. First, we

consider the priority information "0000_1000_0000_0000". This array means the cell can use the first 12 time slots. As we have discussed above, the cell would be scheduled into the 8^{th} time slot, so this cell obtains a schedule within the limit of resources. The priority information would have no affect on the schedule results. On the other hand, if the elementary scheduler receives such an array of priority information as "0000_0000_0100_0000", the result will be different. This priority information limits the time slots available to the cell to the first seven, while there is no schedule for it until the 8^{th} time slot. Thus, even though there are plenty of schedule resources are available from the 8^{th} to 12^{th} time slots, the cell is still going to be discarded because its low priority prevents it from using them.

3.1.4 Operation of the Schedule Register

As mentioned above, the output scheduler should generate an input address for each output of the switch matrix to set up a data path for the cell. In order to generate the input address, we need a copy of schedules. Towards that end, one copy of the schedule results are sent to the input status register, which contributes to update the input status arrays stored in both the input status register and input port controller. The other copy of the schedule array is directed to a sixteen-bit schedule register. Each bit of the register corresponds to a time slot. Since each bit of this register obtains the schedule state from a comparison unit, the contents of this register unit should correspond to the same time slot as the comparison.

The schedule register is used here to record the schedule history of the elementary scheduler. In other words, it indicates which time slots have been scheduled by this

elementary scheduler. As we have mentioned, each elementary scheduler corresponds to one input-output pair. When a schedule is made in an elementary scheduler, it means that a cell is scheduled to be switched between this input-output pair. Indeed, the states in this schedule register indicate in which time slot this input-output pair will switch a cell.

Clearly, the schedule array is the accumulation of the results of the schedule array that come from the comparison units. We should use the schedule array to update the states in the schedule register. Towards that end, an OR gate is used. What is written into the schedule register is the logical OR of schedule array and the previous state in the schedule register. Since the new state always relies on its previous state, all the states in the schedule register should be set to *0* when the scheduler is powered up, which means no schedule has been made. When a new schedule, a logical *1*, appears in the schedule array, it will modify the state in the corresponding bit of the register. If no schedule comes in, the states in the schedule register are kept intact.

Furthermore, as stated above, the shift register is adopted to indicate the scheduling history of the following 16 time slots, so the binary array stored in the register should be shifted one bit per time slot so that the state in the register can correspond to the proper time slot. Hence, the shift register is shifted one bit per time slot. The last bit which corresponds to the 16^{th} time slot is loaded with a *0*, and the state stored in the first bit of the register is loaded into another shift register, as shown in figure 15. This register is also a parallel-in-series-out shift register, whose function is to shift the input address out of the output scheduler.

3.1.5 Input Address Generation

As mentioned above, the purpose of keeping a copy of schedule array in the scheduler register is to generate the input address to be passed to the switch at the correct time when a cell is to be switched. In this section, we will discuss how to generate the input address from the schedule arrays. To solve this problem, we should put the elementary scheduler into a column of elementary schedulers, which is sketched in figure 16. Each elementary scheduler in this column includes a shift register unit and they are connected together into a 16-bit shift register. This shift register achieves the schedule result from the first bit of the schedule register in the elementary scheduler in parallel and shifts them out to the switch matrix. What is loaded into this shift register is just the input address for the cell to be switched in the current time slot. This address is sent to the corresponding output port of the switch matrix to select the input.

Recall that each column of elementary schedulers corresponds to an output, and each row of elementary schedulers corresponds to an input. Also, in the elementary scheduler each comparison unit corresponds to a time slot. Now let us consider the 16 comparison units from different rows, which correspond to the same time slot, say the first time slot. They are highlighted in figure 16 by a box. In this module there are sixteen single-bit shift registers, which indicate whether there is a schedule made for its corresponding input port. Thus, if we associate this array of schedules together, it just indicates which input port is scheduled to send the cell to this output port. For example, such an array as "0000_0000_0001_0000" is stored in these 16 registers corresponding to the first time slot. This array shows that there is a

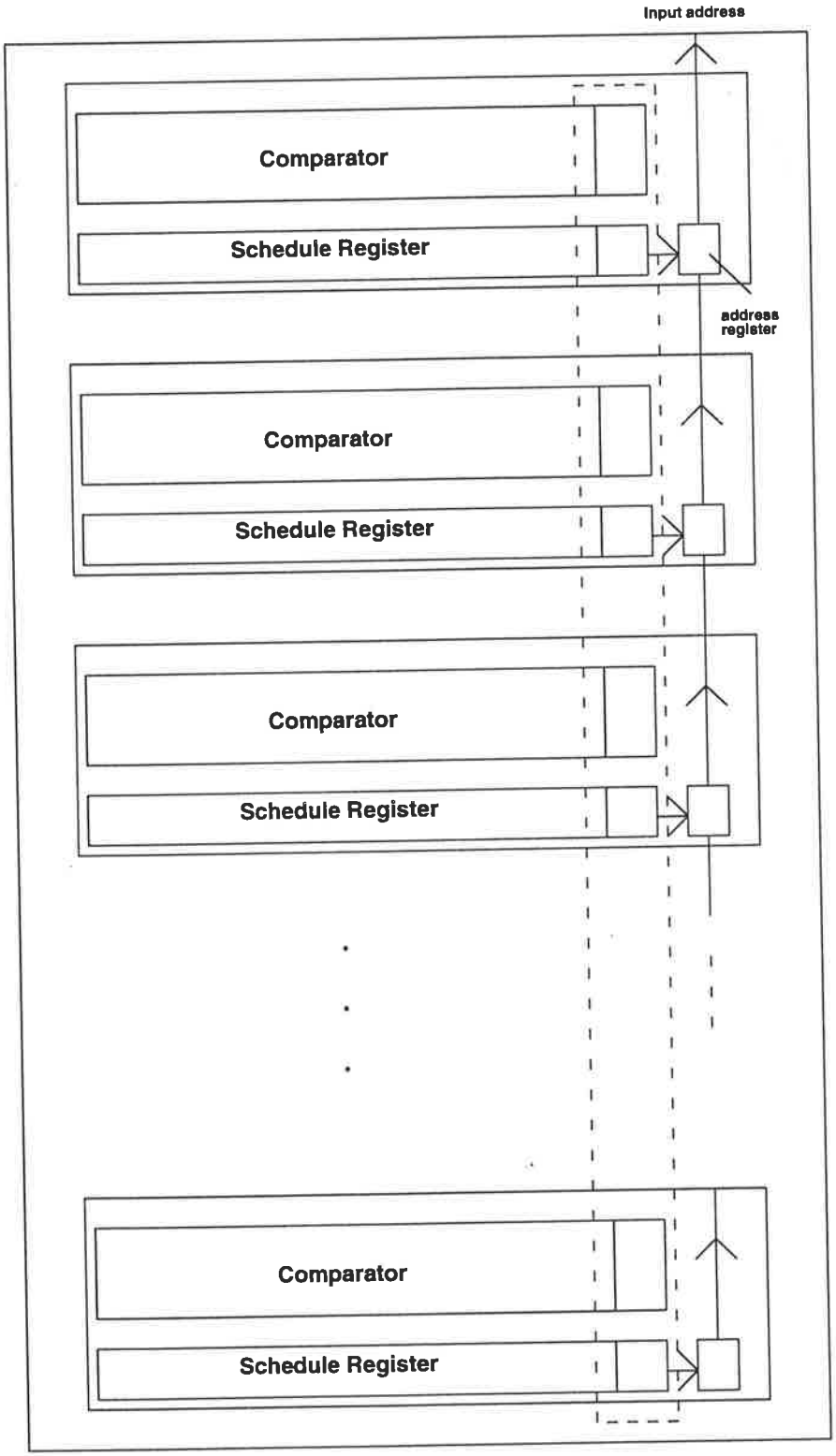

Figure 16 A column of elementary scheduler

schedule made for the 9th input port, so consequently the input port controller corresponding to the 9th input port would send the cell to the switch matrix at the beginning of next time slot. The array of schedules provide switch matrix information to the output port, which then uses this information to select the appropriate cell from the input port. The array of schedules effectively provides an input address to the output port of the switch matrix.

Some words are worthwhile to explain that a cell and its corresponding input address always arrive at the switch matrix in the same time slot. As mentioned above, the input address is generated according to the schedule. When a schedule is generated it is written into one bit of the schedule register that corresponds to the scheduled time slot. Recall that the other copy of schedule result is sent to the input port controller, which helps to write the cell into a memory unit that corresponds to the scheduled time slot. After each time slot, the state in the schedule register is shifted by one bit so that each bit corresponds to its preceding time slot. Similarly, the pointer in the main buffers shifts one bit after each time slot and all the cells stored in main buffers correspond to their preceding time slot. When the scheduled time slot becomes the current time slot, the schedule result stored in the schedule register is loaded into the shift register, which acts as the input address. At the same time, the pointer in the main buffers moves to the memory unit that stores the corresponding cell of this schedule and the cell written into a temporary buffer.

As it takes one time slot to shift the input address in series into the switch matrix, the cell from the main buffers have to be stored in the temporary buffers #4 for one time slot. At the beginning of next time slot, the input address is ready in the selector slice

of the switch matrix and they are loaded into the multiplexer. Simultaneously, the cell is sent to the switch matrix. Therefore, a cell and its corresponding input address always arrive at the switch matrix in the same time slot.

3.1.6 Circuit Design of the Comparison Unit

In order to understand the operation of the comparison unit, we have to step to a lower level, namely, circuit level, to see how the status arrays are compared.

The circuit of the comparison unit and its blocking circuit are shown in figure 17. From the point of view of the inputs *i* (input status) and *x* (output status), there are three paths and each path takes charge of a particular function. Specifically, path 1 will compare the input and output status and generates the schedule result; path 2 will compare the input and output status and update the output status array conditionally; path 3 takes charge of generating the interface signal to the subsequent unit.

3.1.6.1 Generating the Schedule

The path labelled 1 is to generate the schedule. The operation can be readily described with the following equations:

$$e = i \; NOR \; x$$

$$e\_tmp = e \; AND \; up'$$

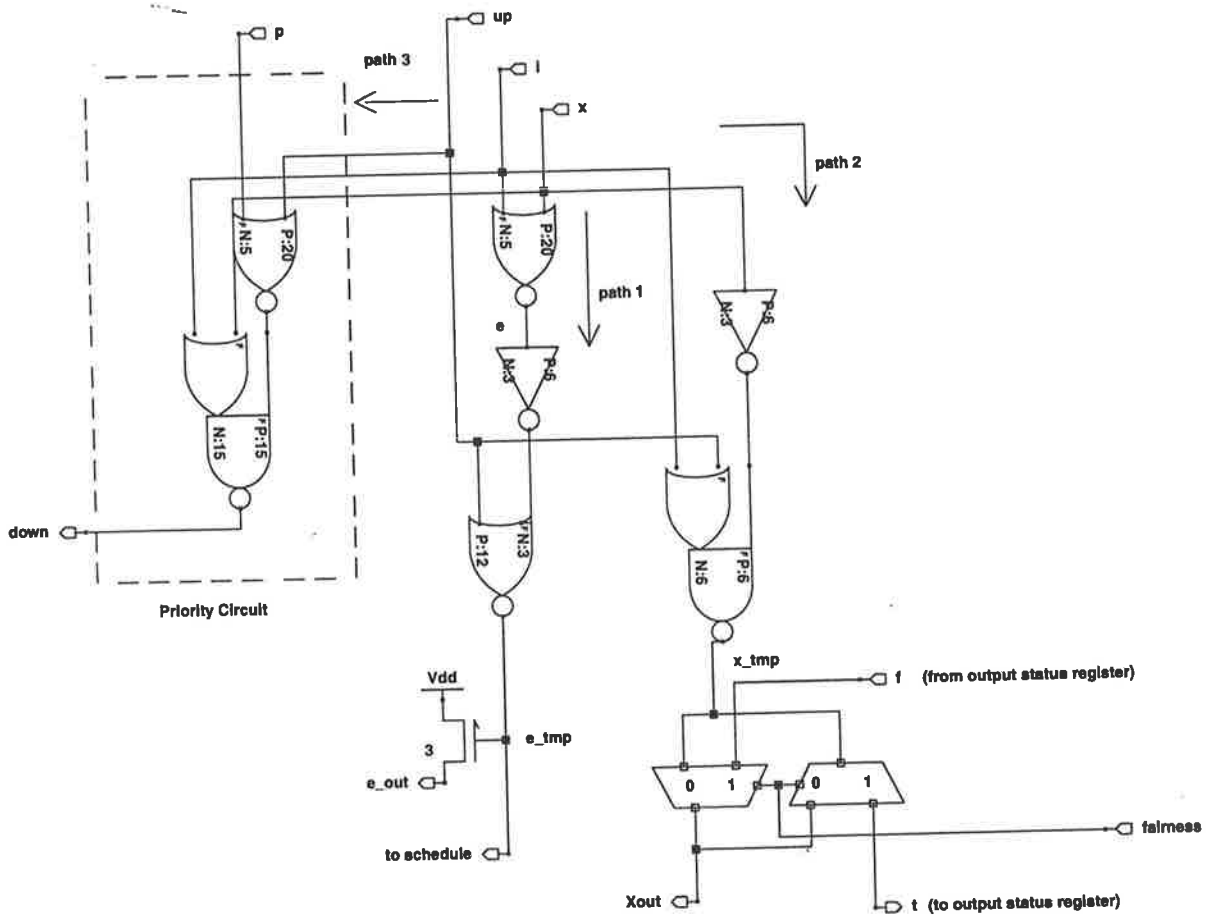

Figure 17 Circuit of comparison unit

The purpose of the first equation is to compare *i* and *x*. Specifically, when both the input status *i* and output status *x* are *0*, which means both input and output port are available, the result *e* (schedule candidate) will be *1*. That means this time slot is suitable to make a schedule. However, it is only a candidate, whether it can become a real schedule or not is dependent on whether this comparison unit is activated. As we discussed above, for the first comparison unit, the enable signal comes from the output address and for the other units the enable signal comes from the interface signal of the upper unit. We employ a logical AND between the *e* and *up* (interface signal from upper unit) and their result is a real schedule result. For example, when the *up* is *1* that means this comparison unit is disabled, the *e_tmp* will definitely be *0*, no matter what the signal *e* is. When the *up* is *0*, then *e_tmp* would follow the value of *e*. Therefore, if *up* is *1*, a schedule candidate (*e=1*) will become a schedule. Note that the signal *e_tmp* has two branches: one is connected to the schedule register and the other directs to an NMOS transistor. For the first case, we have described the reason of why we should store this schedule result. Now let us study the other branch.

In the previous section we have explained that what the input status register receives is the result of a logical OR of sixteen schedule results that correspond to the same time slot but different output ports. This transistor is used here to realise the wired OR. Note that the "gate" of this N-transistor is connected to the schedule signal; the "source" is connected to the Vdd; the "drain", *E_out*, is connected to a bus. This bus connects the input of the input status register with 16 schedule result outputs corresponding to the same time slot and different output ports. When the *e_tmp* is a logical *1*, the transistor is turned on. The "source" Vdd will charge the bus through

the transistor and pull it up to *1*; on the other hand, if the *e_tmp* is a *0*, the transistor will be turned off and output a high impendence that does not affect the state of the bus. In order to realise a logical OR of these 16 schedule results, we should discharge the bus to logical *0* at the beginning of each time slot. If one or more schedule results connected to this bus is *1*, they will pull the bus up and the input of the input status register would be a *1*. Alternatively, when none of these 16 schedule results are *1*, all of them will output high impedance and the bus will keep the state *0*. Thus the input status register will achieve a *0*.

We can also use truth table to explain the operation:

i	x	up	e_tmp	e_out
-	-	1	0	Z
0	0	0	1	1
0	1	0	0	Z
1	0	0	0	Z
1	1	0	0	Z

Where Z' represents high impedance, and '-' represents "don't care".

Table 1 Truth table of schedule generation

3.1.6.2 Updating the Output Status

Since we have understood how the schedule result is made, let us go on to study the second path, which is to compare the input status and output status and update the

output status, if a schedule is made. The operation of this path can be described with the following equation:

$$x\_tmp = (i' \; AND \; up') \; OR \; x$$

Where i' represents the inverted i, up' means inverted up and x_tmp represents the new output status. This equation informs us that when the enable signal up is a logical 1, in other words, this comparison unit is not requested to work, the result of (i' AND up') would certainly be 0. Consequently, x_tmp will follow the previous value of output status x. In contrast, if this elementary scheduler is activated, namely, $up=0$, then last equation can be rewritten like this:

$$x\_tmp = i' \; OR \; x$$

The reason is that when $up=0$, the result of (i' AND up) will always be decided by the value of i'. When i is equal to 1, this means the input port has been scheduled by another cell, so no schedule can be made and x_tmp will follow the previous x. If this time slot for this input has not been scheduled out, the signal i would be 0. In this case, x_tmp is equal to 1, which means a schedule will be made and the output status is updated to 1.

The function described above can also be described with a truth table.

up	i	x	x_tmp
-	-	1	1
0	0	0	1
0	1	0	0
1	0	0	0
1	1	0	0

Where '-' represent "don't care".

Table 2 Truth table of output status updating.

Here we have achieved the updated output status, *x_tmp*, next we discuss how this signal is distributed. Recall that in order to defeat unfairness, the output status arrays are loaded into the different elementary scheduler in each time slot and consequently the updated output status arrays are returned to the output status register from different elementary schedulers in each time slot. This distribution circuit is employed here just for realising these functions.

As shown in figure 17, the distribution circuit is a combination of a 2:1 demultiplexer and a 1:2 multiplexer. The signal *x_tmp* has two branches of output: one is *Xout* and the other is *t*. Clearly, we need a demultiplexer here to decide which branch the signal should be directed. On the other hand, as we see in the circuit diagram, the signal *Xout* is driven by two sources: one is from the *x_tmp* and the other is the output status from the output status register, *f*. In order to select one of

them we need a multiplexer here. Both the multiplexer and the demultiplexer are controlled by the signal labelled *fairness*, because all of these operations are to defeat unfairness.

When a row of elementary schedulers receives a logical *1* from signal *fairness*, (see figure 17) that means the next row of elementary schedulers will act as the entry-scheduler for the output status array. For this case, the output *Xout* should be connected to the output status from the output status register. The other source of the *Xout* from *x_tmp* should be latched. The output status goes through all the elementary scheduler in a column will go back to the output status register from the entry-point. Hence, the signal *x_tmp* is connected to the output that directs to the output status register. From the figure 17 we find that when the signal *fairness* is a logical *1*, it will turn on the pass transistor between the *f* and the *Xout* and the pass transistor between *x_tmp* to *t*.

Also, we can describe it with a truth table.

fairness	**Xout**	**t**
0	**X_tmp**	**Z**
1	**f**	**X_tmp**

Where Z' represents high impedance.

Table 3 Truth table of generating fairness signal.

On the other hand, for the common case the elementary scheduler does not act as the entry-scheduler, so it just receives the output status array from its upper elementary scheduler, and sends it to its next one. It is not aware of the existence of the output status register. For this occasion, the signal *fairness* is a logical *0* and it develops a path between the *x_tmp* and *Xout* (see figure 17). At the same time, this signal latches the signal from the output status register and output high impedance to the output status, which makes this elementary scheduler neither modify the states of the output status array, nor be affected by the status array. This completes the description of how the output status array takes part in comparison and how it is distributed.

3.1.6.3 Interfacing with Subsequent Unit

Finally, let us analyse the blocking circuit that interfaces with the comparison unit corresponding to the next time slot. The blocking circuit takes charge of informing its subsequent unit whether they are requested to compare the status array. If the interface signal *down* is a *0*, its subsequent unit is enabled to compare the status array. In contrast, if *down* is a *1*, the subsequent units will be disabled. There are three factors that may affect the value of the interface signal, that is *up*, *p* and the comparison result of *i* and *x*. Therefore, the blocking circuit needs these signals as inputs. (see figure 17)

The operation of the blocking circuit can be described with the following equation:

$$down = (i \; NOR \; x) \; OR \; p \; OR \; up$$

where *up'* represents the inverted *up*.

This equation shows the contribution from these three factors clearly. If this elementary scheduler is disabled, namely, *up=0* then signal *up'* will be *1*. No matter what is the comparison result of *i* and *x*, signal *down* will certainly be *1*. Then the subsequently unit will be disabled. If the signal *up* is a *1*, the interface signal *down* is decided by the other factors. If the signal *p* is a *1*, which means the priority information prevents the cell from using the subsequent time slot, the signal *down* will also be *1*. Thus, the subsequent unit will be disabled. If both *p* and *up'* are *0*, the signal *down* is decided by the comparison result of *i* and *x*. When both of them are *0*, this time slot will be scheduled. Since a schedule has been found, it is not necessary for the subsequent unit to do anything. Therefore, signal *down* passes a *1* to the subsequent unit to disable it. On the other hand, if no schedule is achieved in this comparison unit, namely (*i* NOR *x*) is equal to *0*, then the signal *down* will be *0*. It informs its subsequent unit to go on comparing the input and output status arrays.

The functionality of this part of circuit can also be described by the truth table.

i	x	p	up	down
-	-	-	1	1
-	-	1	0	1
0	0	0	0	1
0	1	0	0	0
1	0	0	0	0
1	1	0	0	0

Where '-' represent "don't care".

Table 4 Truth table of Generating interface signal.

3.2 Input Status Register

In the output scheduler, the input status arrays are maintained and updated in the input status register. We have discussed the basic function of the input status register in the previous section. In this section the structure and operation of the input status register will be discussed in more details.

3.2.1 The Structure of Input Status Register

Figure 18 shows the diagram of an input status register. The input status register has 16 inputs which receive the schedule array from the elementary schedulers. There are 16 outputs that send the input status array to the elementary schedulers.

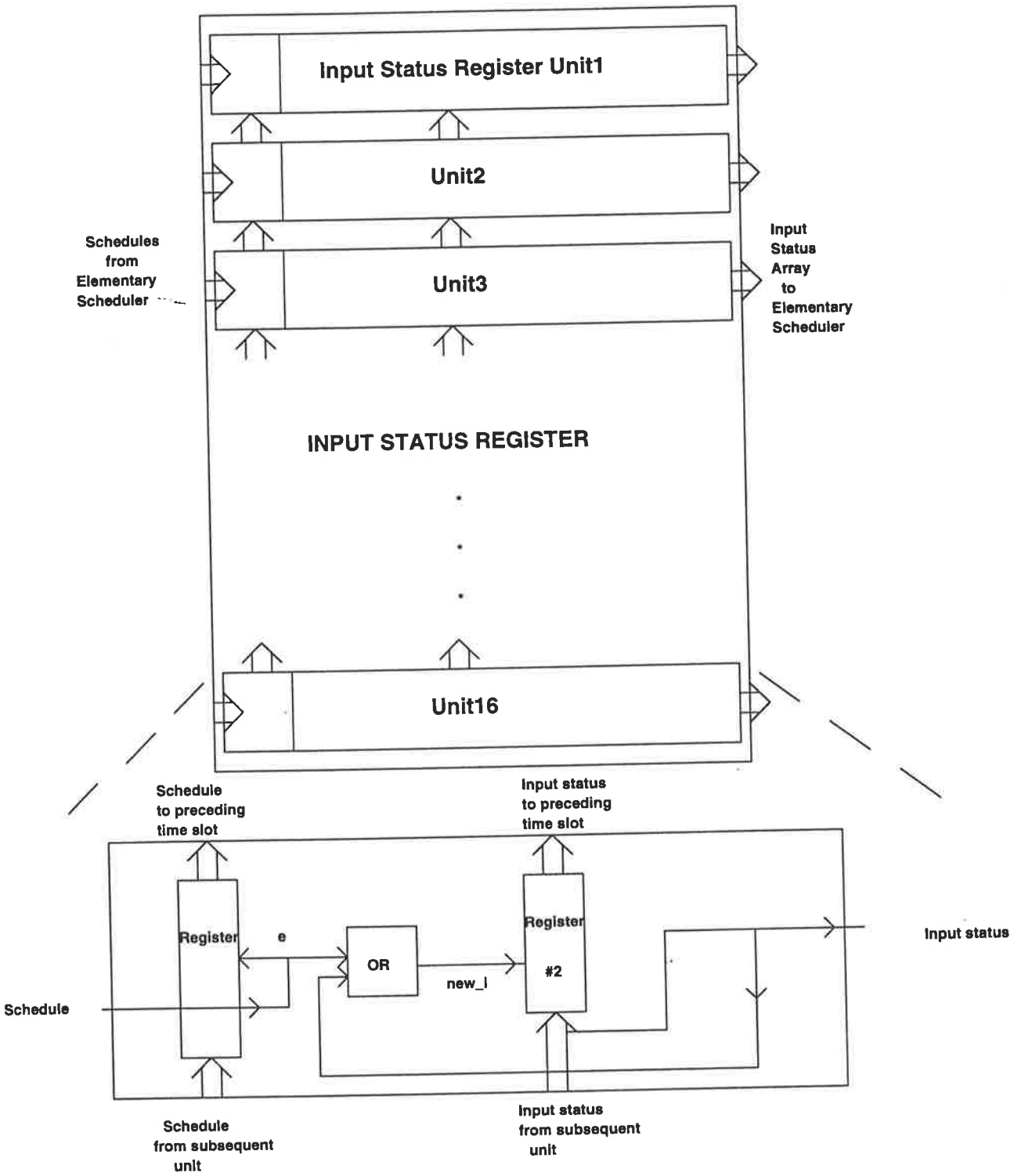

Figure 18 Diagram of input status register

The input status register is comprised of 16 input status register units. Each unit corresponds to a fixed time slot. Recall that in the main buffer, each memory unit corresponds to a floating time slot. Here, each input status register corresponds to a particular time slot. For example, the 1st unit corresponds to the first time slot, the 3rd unit corresponds to the third time slot and so forth. As shown in the diagram, each unit receives a schedule bit and outputs an input status bit.

The diagram of an input status register unit is shown at the bottom of figure 18. Basically, each input status register unit contains two parts: one is the schedule status shift register, the other is the input status register. The schedule status register in each unit is connected in series into a 16-bit shift register. The input status register is comprised of an OR gate and a register. The OR gate is used to change the schedule status into the input status, as explained below. The register is employed to store and shift the input status array.

3.2.2 Operation of the Input Status Register

First of all, let us look at the operation of the schedule status shift register. This 16-bit shift register stores the 16-bit schedule array in parallel at the beginning of a time slot. This 16-bit wide array is shifted out to the input port controller in series. This schedule array is used to place the cells into the buffers in the input port controller as explained in a previous section.

Secondly, we will concentrate on the discussion of input status register. Refer to the diagram of the input status register unit shown at the bottom of figure 18.

From the diagram, we note that the register in each unit is connected one by one. They comprise a 16-bit shift register. The shift register forwards the state in it one bit per time slot, namely the state in the 16^{th} unit is loaded into the 15^{th} unit, the state in the 3^{rd} unit is loaded into 2^{nd} unit and so on. The reason for this operation is straightforward. As mentioned above, each unit of the input status register corresponds to a fix time slot. After each time slot, each bit of the input status array should correspond to a new time slot. Therefore, in order to keep each bit of the input status array always corresponds to the correct time slot, they should be moved into the register units that correspond to the appropriate time slot. In this input status register, the top unit is designed to correspond to the first time slot, and the bottom one corresponds to the 16^{th} time slot. After a time slot, each bit of the input status array should correspond to its preceding time slot. Therefore at the beginning of a time slot, each bit of the input status array is shifted into the upper unit. And the 16^{th} bit is fed a *0* that represents a new time slot to be scheduled (see figure 18).

Note that besides the state loaded into the register, there are another two branches for the input status. One of them is sent out of the input status register. This copy of input status is sent to the elementary schedulers and will be compared with the output status.

The other copy is sent to an OR gate. Recall that the input status register receives the schedule array, but maintains and sends out the input status array. Therefore, we should change the schedule status into the input status before it is stored in the register. This OR gate is used to change the schedule status into input status. If the

previous state of the input status is the logical *0* and a schedule is made for this time slot, namely a logical *1* is received, then the logical OR of these two states is *1*. This state is written into the register, which updates the previous state in the register. If the previous status is the logical *0* and no schedule is made for this time slot, namely a logical *0* is received, the input status will keep its previous value, logical *0*. If the previous value of the input status is logical *1*, which means this time slot has been scheduled out, it is impossible to make a further schedule. Therefore the schedule should be *0* and the input status is still *1*.

3.3 Output Status Register

As the name implies, the out status register is used to maintain and drive the output status array. A diagram of the output status register with the detailed structure of the register unit is show in figure 19. Indeed, the output status register is a simplified version of the input status register. The output status register needs not shift anything out, so there is no shifter register for shifting the schedule array out. Moreover, the output status register receives and stores the output status array and maintains the output status array, so the output status is written into the register directly. Except for the above difference, the operation of the output status register is identical to that of input status register.

3.4 Clock Generator

The clock generator acts as the generator of all necessary signals for the operation of the elementary schedulers and the status registers. A diagram is shown in figure 20.

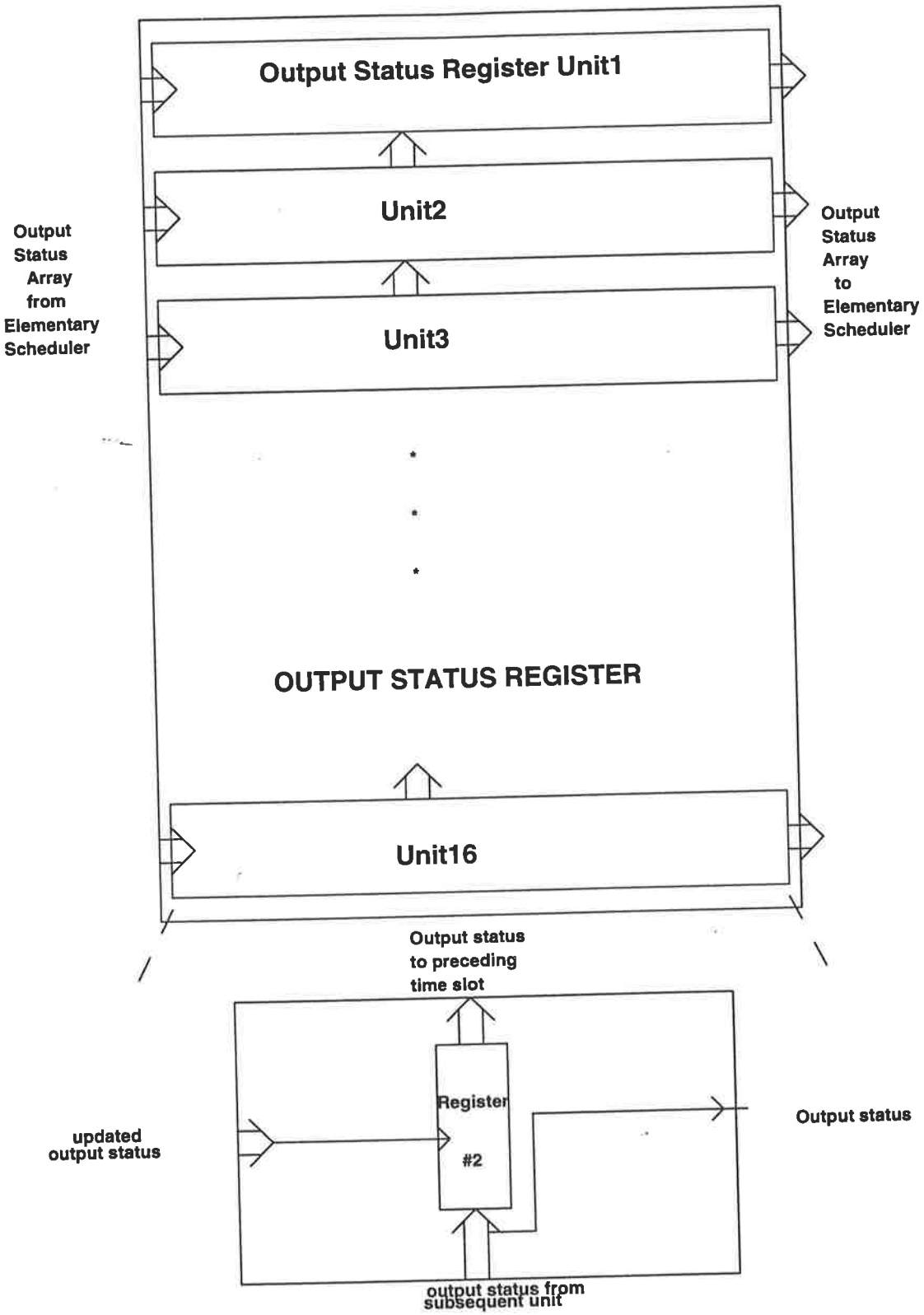

Figure 19 Diagram of output status register

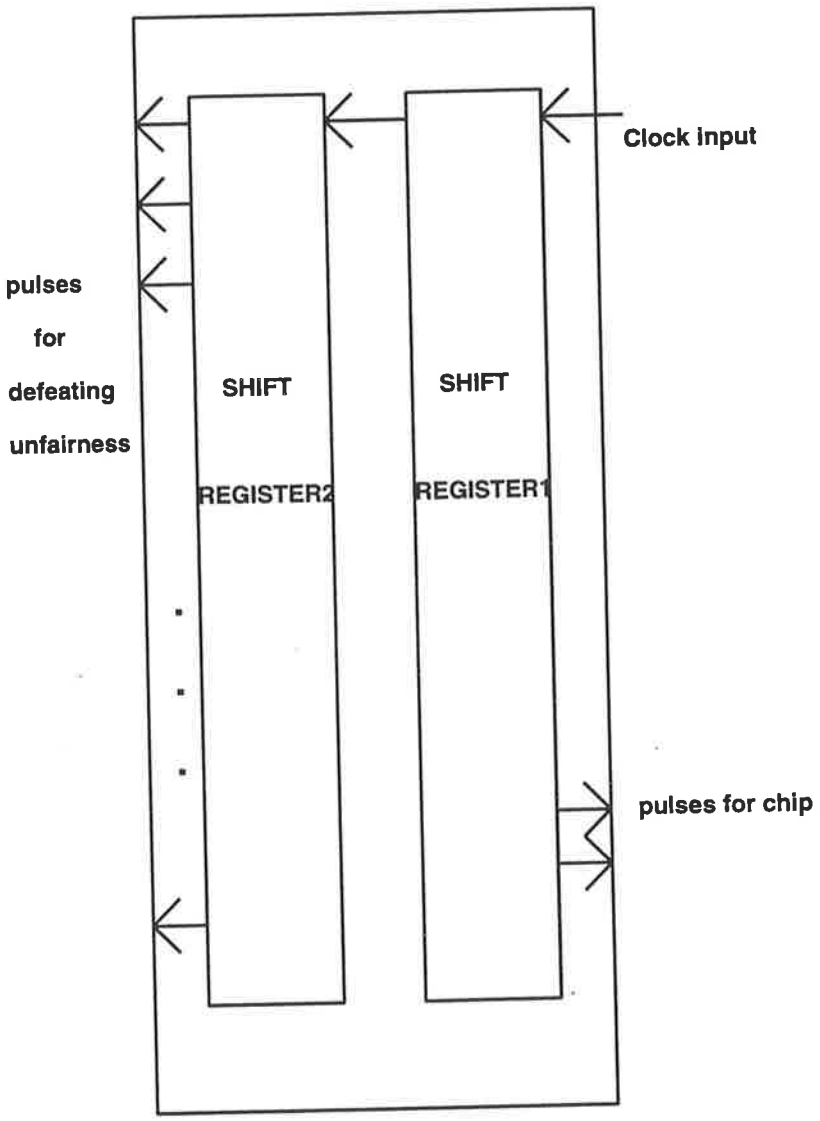

Figure 20 Diagram of clock generator

Basically, it is comprised of two identical 16-bit shift registers. Both of them are set to such a state as "0000_0000_0000_0001" when the chip is powered on. Moreover, the first bit and the last bit of either register are connected so that the states in it can be rotated round and round. Indeed, both of them operate like a pointer used in the read controller of input port controller.

The shift register #1 is used to generate the assertion pulse, which is necessary in the operation of the input status register and elementary scheduler. For example, the input status register needs an assertion signal at the beginning of each time slot so that the input status array is forwarded one step. This signal can be realised using the shift register #1. This register is driven by the clock signal, which asserts 16 times per time slot (we will define the clock frequency in the later section). At the beginning of a time slot, the first bit of the shift register #1 is set to 1; all other bits are set to 0. The output of the first bit of the shift register #1 will be the assertion signal that we need. Using this register, we can achieve any assertion pulse that we need.

A timing diagram of the signal generator is shown in figure 21. The shift register #1 is driven by the clock signal, so the set bit shifts once every clock cycle. We use the output of the shift register #1 to control operation of the whole scheduler. In addition, the signal A1(1^{st} bit of shift register1) is used to drive the shift register #2, which generates a signal for defeating unfairness. As shown in the diagram, A1 is asserted at the beginning of each time slot. When A1 is asserted, it drives the shift register2 to shift one bit.

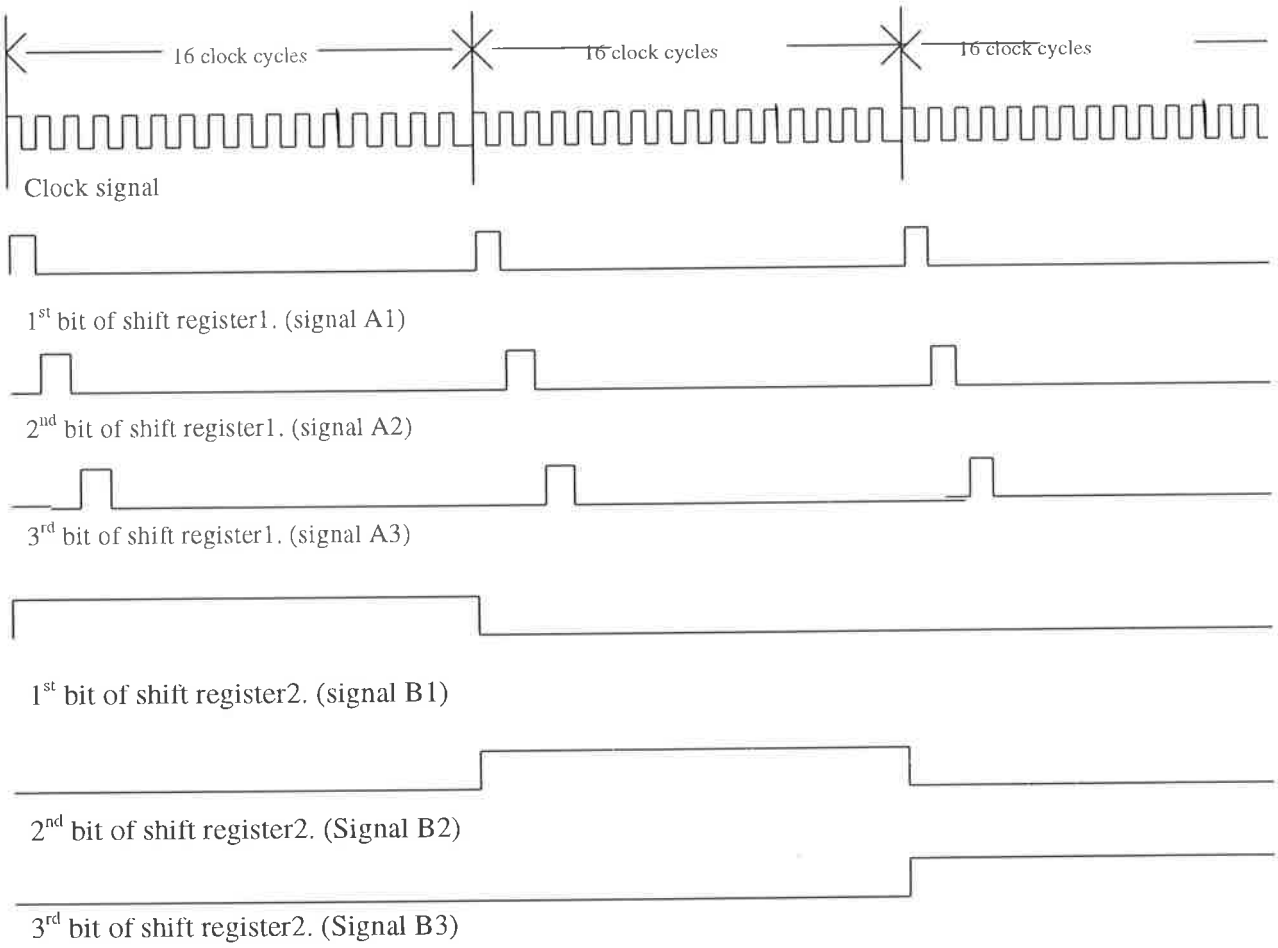

Figure21 Timing diagram of signal generator.

Chapter Four

Physical Design of the Output Scheduler

In the previous chapters, we discussed the operation of the output scheduler in detail. As we mentioned in the first chapter, this output scheduler is designed to the circuit level, so this chapter we will discuss some VLSI (Very Large Scale Integrated circuit design) circuit design issues.

We will discuss three topics in this chapter. First of all, the design methodology is illustrated, which shows the design flow of this output scheduler. Secondly, we will discuss some techniques used in the VLSI design to achieve high speed, comparatively low power dissipation and reasonably compact area. Then, we will give the simulation results of this output scheduler.

4.1 The Design Methodology

The design of this output scheduler follows a top-down methodology, which is widely used in modern ASIC (Application Specified Integrated Circuit) design [8] [9]. Specifically, it includes the following steps:

- **High-level design specification and partition**: According to the role of the output scheduler to be played in an ATM switching system and the time scheduling algorithm, we specify the function of the output scheduler. Then this function is divided into a few basic functional blocks to realise it. As stated

above, the output scheduler is divided into 4 basic functional blocks, namely elementary scheduler, input status register, output status register and clock generator.

- **Develop and validate functional models**: In this design, the models of the output scheduler are developed with VHDL (Very high speed integrated circuit Hardware Description Language). We use VHDL to describe the behaviour of each basic functional block and combine them together according to the architecture defined in the last step. The models are simulated to make sure that they can realise the functions that we expect. Basically, modelling has three functions: firstly, it can verify the correctness of the design specification at high-level abstraction; secondly, the models can be used as the source for synthesis; thirdly, the stimuli that will be used for functional verification can be generated from the models.
- **Schematic design and simulation**: In order to achieve the best performance, the schematic of the scheduler is designed manually, instead of being synthesised from the VHDL code. The schematic is simulated to the switch level to ensure the correctness of its function. The schematic design can not provide such information as the RC delay due to the contacts and wires, so we can not achieve any accurate simulation result. Therefore, switch level simulation is carried out, which ignores the non-linear effects of the transistor. It only shows the functional validity. We can use the stimuli generated from VHDL models to simulate the circuit automatically.
- **Layout design and simulation**: The layout is also drawn manually to achieve high-speed, low power and small area. The layout is designed on the basis of the schematic that is developed and validated in the last step. Since the layout has

included all the delay information, we use Hspice to conduct full-analog simulation, from which we can achieve accurate delay and power dissipation estimates. As Hspice considers the non-linear effects in the operation of the transistor, it is very accurate but expensive in calculation time. Therefore, it is not feasible to simulate the whole chip or very large-scale circuit. Usually, Hspice is used to analyse and simulate the critical path of the circuit.

- **Functional Verification**: In order to guarantee that the circuit can realise the functions we expect, we use the switch-level simulator again to verify the function of the circuits. Similarly, we can use the stimuli generated from the VHDL codes to verify the function.
- **Layout Verification**: Layout verification includes two steps: LVS (Layout Vs Schematic) and DRC (Design Rule Check). Since the layout is designed on the basis of the schematic, it makes sense that they have identical netlists. Therefore, we should compare the layout and the schematic. As we know, the final fabrication of the circuit on the wafer relies on the layout of the circuit. For each technology, there are some specific electronic rules for the circuit layout. If some rules are violated, it may cause some malfunctions of the circuits. Hence, a DRC of the overall circuit is necessary.

Compared with the bottom-up design methodology, the most significant advantage of top-down design is earlier error detection. As stated above, the design begins with the high-level abstract modelling from which we can detect errors before any detailed design is begun. After we make sure that the models are correct, we step to the lower level, schematic design. Similarly, the layout design is not carried out until the schematic is proved to be correct. Thus, when the circuit is finally sent to

fabrication, the probability of error will be very low. Earlier error detection results in the reduction of the development costs and increases the chance of first pass success.

In this section, we introduced the methodology of ASIC design. With a general understanding of ASIC design flow, we will discuss some specific design topics in the next section.

4.2 Techniques for High Performance Digital Design

In the second chapter, we discussed the operation of the output scheduler in the domain of functional description. Referring to the design flow introduced in the last section, we have finished the schematic design. In the next few sections, we will discuss some issues on layout design, because the layout will decide the final performance of the chip.

4.2.1 Design Specification

Before we start to discuss the particular design issues, we should specify some parameters.

The scheduler is driven by a clock whose frequency is 400 MHz. In other words, the period of each clock cycle is 2.5 ns. This value is decided according to the design objective. Recall that the output scheduler should be designed to support a 10Gb/s/Channel switch matrix. For each ATM cell, there are 53 octets, namely 424 bits. For the switch matrix that can support 10Gb/s/Channel, it takes 42.4 ns to

switch an ATM cell. In order to support such a high-speed switch matrix, the scheduler has to finish one scheduling process within 42.4 ns. We have defined the time for switching one cell as one time slot. Therefore, for convenience we specify each time slot to be 40 ns. As defined in the last chapter, the 16-bit output address, schedule array and input address should be shifted into or out of the scheduler in series within one time slot. Clearly, each time slot should include 16 clock cycles. Therefore, the period of each clock cycle should be 2.5 ns. In order to simplify the design, the scheduler is driven by a single-phase clock signal.

The target process technology of the output scheduler is 0.25um TSMC CMOS technology with 5 metal layers and single poly. The power supply is 2.5 V.

4.2.2 Design Requirements

Basically, the quality of a VLSI chip is decided by three factors, namely speed, power dissipation and area. Just like many other engineering fields, it is impossible to achieve the best performance on every factor, because some factors always conflict with others. Therefore, an optimal design is a compromise between those factors.

For example, suppose we are designing the CPU for a PC. For a PC, we need not be too concerned about low power and we can install a fan to cool down the processor. Moreover, we have sufficient room for a processor. Therefore, we can trade the power dissipation and area for the speed. On the other hand, if we are designing a CPU for a portable computer, whose power supply comes from the battery, we have

to minimise the power dissipation of the processor so as to increase the lifetime of the battery. In addition, portability requirements dictate that the processor be designed as compactly as possible. Hence, the power dissipation and area are the key factors to be satisfied.

As mentioned above, the objective of this design is to support the switch matrix that works at a speed of 10Gb/s/Channel. Consequently, the scheduler should finish a scheduling process within 40 ns. In each scheduling process, the output status array should ripple through all the 16 elementary schedulers in one column. It is really a challenge! Therefore, the speed should be a key factor to be considered in the design process. On the premise of a satisfactory speed, we can try to minimise the power dissipation and area. Therefore, all the design decisions are based on this basic principle.

4.2.3 Design for High Speed

As stated above, the output scheduler should finish one scheduling process within 40ns and the requests and the scheduling results have to be shifted into or out of the scheduler within one time slot. All of these require that the circuit must be designed with much care. In this section, we will discuss some techniques used in the layout design of the scheduler.

4.2.3.1 Floorplanning

Floorplanning is the exercise of arranging blocks of layout within a chip so as to minimise the area and/or maximise the speed. Many detailed design decisions are closely related to the high-level topology of the functional blocks, so we discuss the floorplan of the scheduler first. A diagram of the scheduler floorplan is shown in figure 22. In the centre of the scheduler is the clock generator. There is one input status register in each row of elementary schedulers and the register lies in the middle of this row. The shift register for output addresses and the decoder for priority information are integrated into the input status register. There is one output status register in each column of elementary schedulers and the registers are placed in the middle of each column. As shown in the diagram, the input status registers and output status registers comprise a cross that divides the 16x16 elementary scheduler array into four parts. Each part consists of an 8x8 array of elementary schedulers.

That the input status registers and the output status registers are placed in the middle of the row or column where they lie makes the structure of the output scheduler more symmetric. Since the input status register and output status register should exchange information with the elementary schedulers frequently, this symmetric structure helps to reduce the RC delay from the wiring. Also, it is advantageous for power conservation (we will come back to this point later).

The reason for putting the clock generator in the centre of the output scheduler is to simplify the clock distribution. As stated above, the clock generator takes charge of buffering the clock signal and generates the assertion pulse. Each

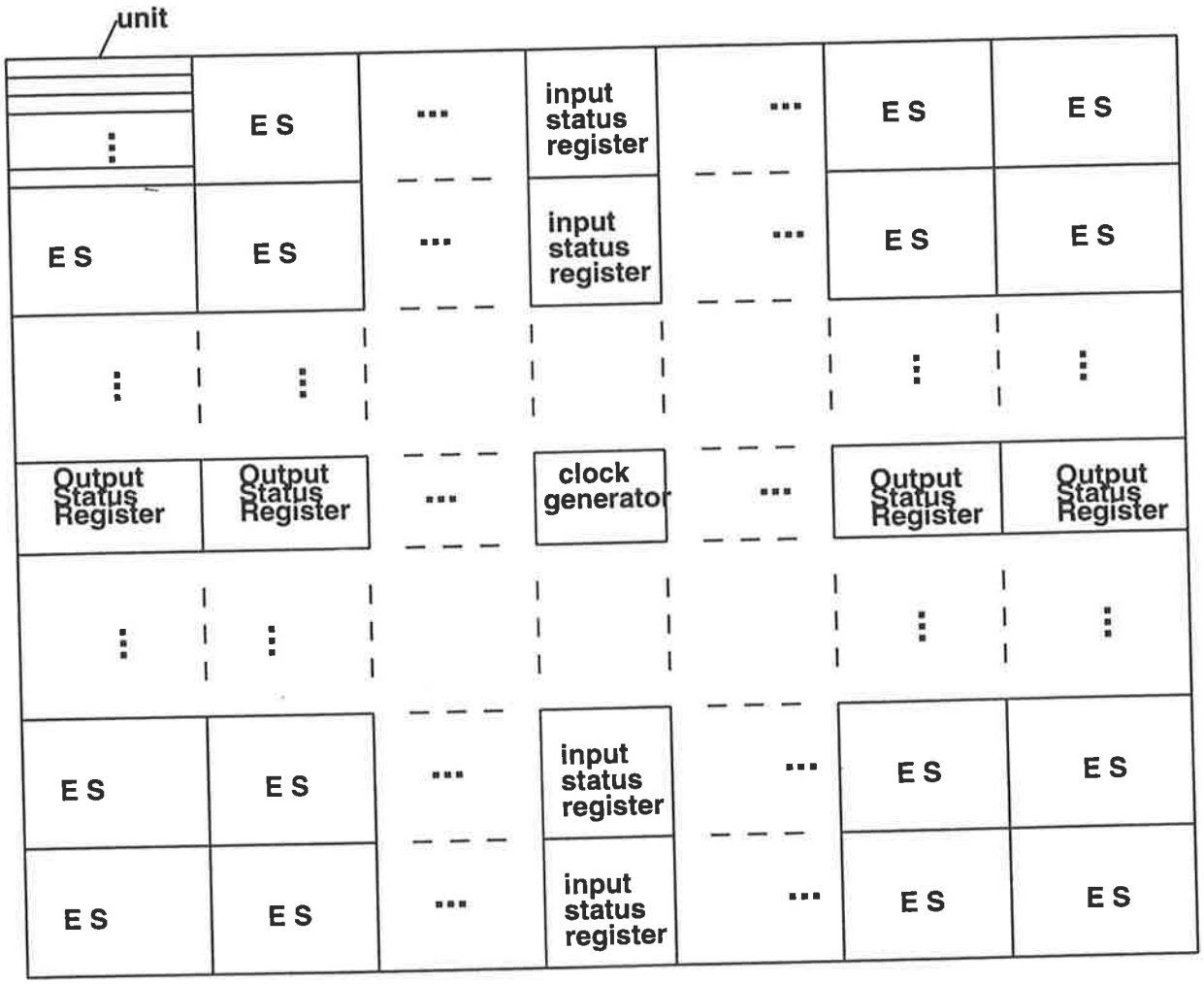

Figure 22 Floorplan of elementary scheduler

functional block needs some assertion pulses or clock signals and all of these signals come from the clock generator. Clearly, when we put the clock generator in the centre of the chip, the assertion pulses and clock signals can reach each functional block with the minimum distance and best symmetry. Also, it is helpful to minimise the skew. We will discuss this topic in the next section.

4.2.3.2 Clock Distribution and Skew

In the output scheduler, both the input and output status registers are sequential circuits. They consist of a large number of registers and latches. All of them should be driven by the clock signal or assertion pulse. This huge fan out acts as a large capacitive load on the clock signals and assertion pulses. The load is further increased by the capacitance from the wire itself, which is distributed from the clock generator to each corner of the chip. Therefore, we need sufficient buffers to amplify the signals and drive them into functional blocks.

A closely related problem is skew. Some wires for clock distribution may reach a length of centimetres. Such long clock wires introduce a substantial series resistance, even if we use the metal layer. A clock line thus behaves as a distributed RC line. As the delay of an RC line is a function of the length, the flip-flop connected to the same clock signal may observe different transition times due to their different distance from the driver. This effect is clock skew [9]. Skew can severely affect the performance of the sequential circuit. In the actual design, it is very difficult to make the skew zero. The most important issue is to limit the skew to that which the circuit

can tolerant. As the output scheduler works at a very high speed, this makes it very sensitive to the skew. Therefore, we have to try to minimise the skew.

A practical way to solve the skew problem is to route the clock signal carefully and use a hierarchical clock-buffering scheme. Figure 23 shows the structure of a buffer tree that is used in the output scheduler. Note that the clock signal is driven by the buffers stage by stage from the signal source to final circuit. Specifically, the signal is amplified to drive the buffers in each column. The buffers in each column drive the signal to the buffers in each elementary scheduler. The buffers in each elementary scheduler drive the signal to each flip-flop. Clearly, this approach does not result in a zero skew, but it decreases the skew substantially. The reason is that the intermediate buffers isolate the local clock nets from upstream load impedances and amplify the clock signals degraded by the RC network. Therefore, the skew is decreased and the signal slope is kept steep.

4.2.3.3 Critical Path Analysis and Optimisation

In a circuit, there are a large number of paths and each path has a characteristic delay. When we talk about the delay of a circuit, what we refer to should be the maximum delay from all these paths. The path with the maximum delay is called the critical path. The critical path will determine the speed of the whole circuit. Therefore, we should analyse the circuit to find the critical path and optimise it to achieve the minimum delay.

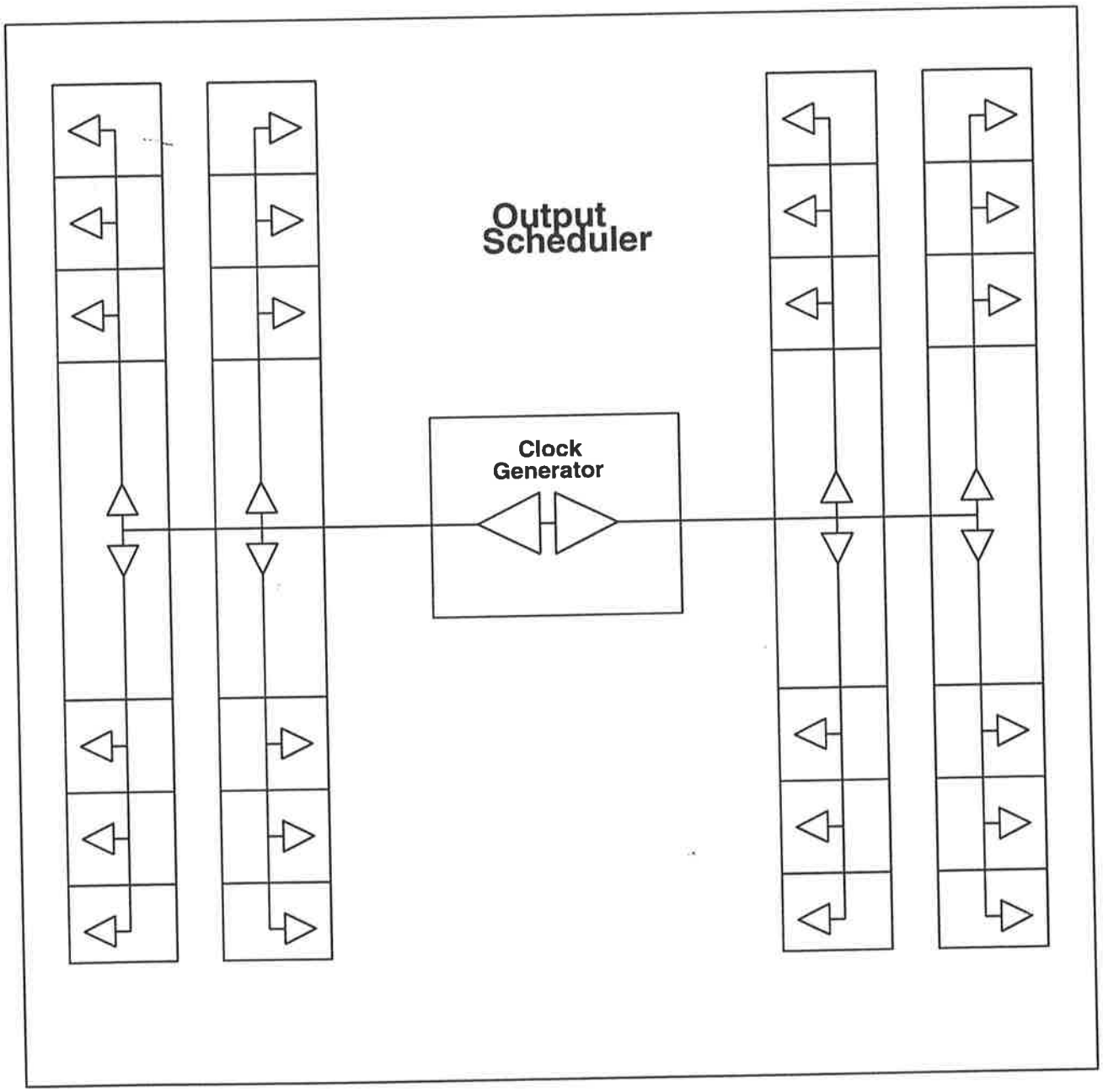

Figure 23 Structure of buffer distribution

- **Identifying the Critical Path**

In the last chapter, we have noted that in the scheduling process the output status array ripples through 16 elementary scheduler in a column and is compared with the 16 input status arrays. The input status arrays and the priority information are sent to each elementary scheduler at the beginning of each time slot, while the output status array goes through each elementary scheduler one by one. This implies that the input status arrays and the priority information will not bring any delay, as they are always waiting for the output status array. Thus, we should only analyse the flow of the output status array to look for the critical path. After the status arrays are compared in an elementary scheduler, the elementary scheduler generates a schedule array and an updated output status array. The generated schedule array is logical ORed with other schedule arrays that are generated by the elementary schedulers in the same row to produce a final schedule array, then the operation in this path is finished. However, the updated output status array will be compared with other input status arrays in the subsequent elementary schedulers of this column. Clearly, the critical path will be one of the paths in the flow of output status array.

A column of elementary schedulers is shown in figure 24. We assume that the comparison begins from the top elementary scheduler and finishes at the bottom elementary scheduler. As we have discussed, in each elementary scheduler the input and output status arrays are compared from the #1 unit to the #16 unit. In the column, the output status array is compared with the input status arrays from the top elementary scheduler to the bottom elementary scheduler as shown in the diagram. Therefore, the maximum delay for scheduling should be the delay from the moment

that the status arrays appear on the #1 unit of the top elementary scheduler to the moment that the #16 unit in the bottom elementary scheduler outputs the output status.

Now let us study the delay quantitatively. Each comparison unit receives the output status from the unit corresponding to the same time slot in the upper elementary scheduler. The unit compares the received output status array with the waiting input status array, then sends the updated output status array to the subsequent unit. We define this delay as t1. Each unit receives the interface signal from the unit in the same elementary scheduler corresponding to the preceding time slot. The unit modifies this interface signal and passes it to its subsequent unit. This delay is defined as t2. Referring to figure 24, we note that there are a large number of paths from the #1 unit in the top elementary scheduler to the #16 unit in the bottom elementary scheduler. With a careful analysis, we find that the delay is identical for all paths. The total delay is

$$\text{Delay} = 15 \times t1 + 15 \times t2$$

For example, in the Path 1 the delay is 15 x t1 + 15 x t2; in the Path 2 the delay is 2 x t2 + 15 x t1 + 13 x t2; in the Path 3 the delay is 15 x t2 + 15 x t1. All of these paths result in the same delay. Therefore, we can conclude that any path between the #1 unit in the top elementary scheduler and the #16 unit in the bottom elementary scheduler can be the critical path. The final delay is decided by the delay from each unit, t1 and t2.

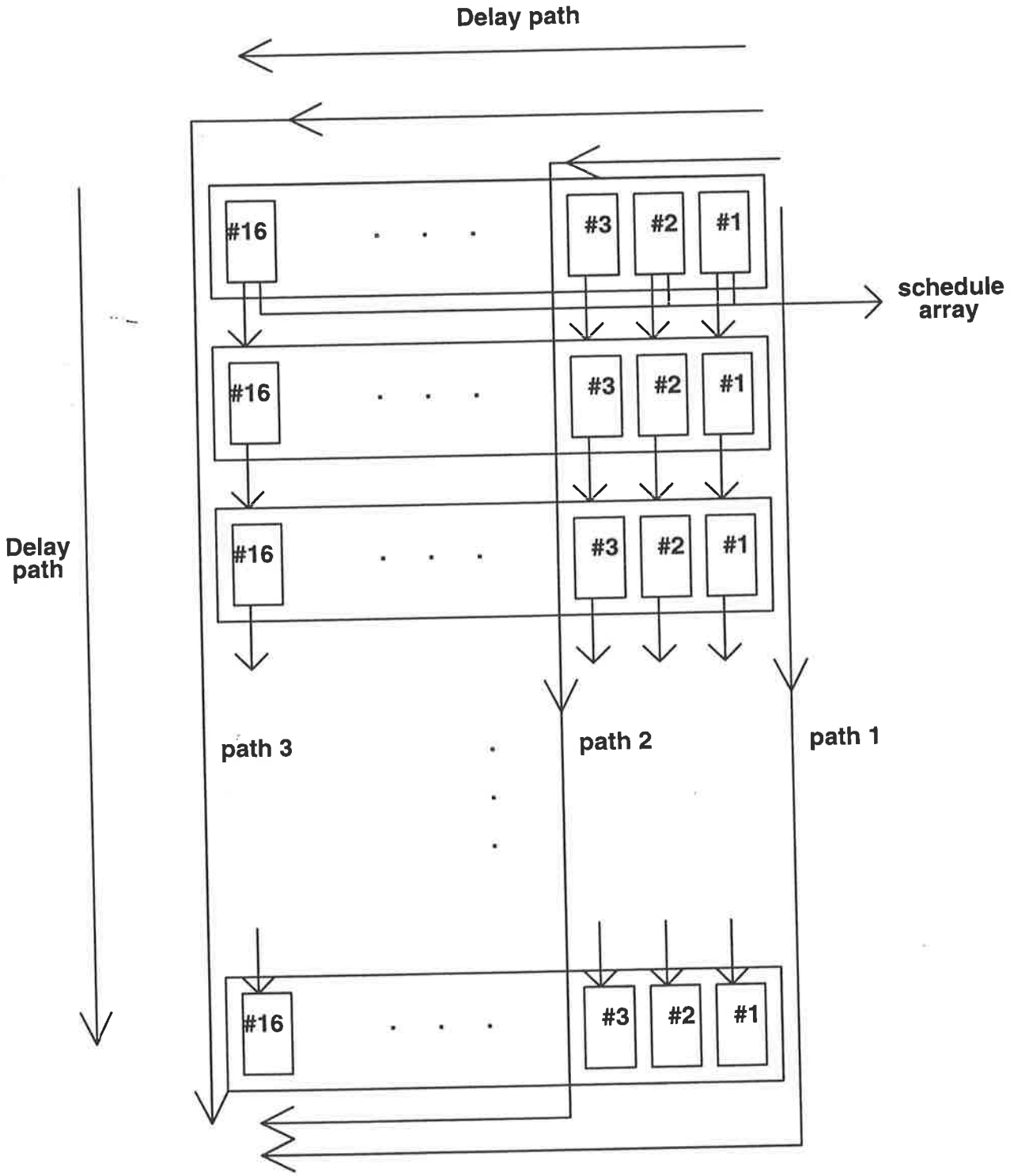

Figure 24 The critical path of output scheduler

- **Optimising the Critical Path**

Basically, we have two ways to optimise the critical path: one is to use lookahead among the comparison units in the elementary scheduler so as to minimise the delay; the other is by careful layout design. In this section, we focus on the layout design techniques, so we leave the first way to be discussed in a later section.

According to the analysis in the last section, the delay of the critical path is determined by the delay from updating output status (t1) and the delay from the interface circuit (t2). As the total delay is 15 times t1+t2, any small decrement on t1 or t2 will have a significant reduction on the total delay.

Design Priority

Recall that in each comparison unit, there are three paths: interfacing with the subsequent unit, updating output status and generating the schedule result. The former two paths are in the critical path, so we should give them a better priority in the layout design to minimise their delay and leave the path of generating the schedule array to suffer more delay. This idea is mainly represented in the placement of layout and the selection of the layers to be used for interconnection.

For example, we want to connect two gates that are in the critical path, but there are some other gates physically between them and the layout of these gate is not "metal transparent". As we know, the metal layer has a very small resistance, so it is the most suitable layer for connection. However, as in this case the metal can not go

through the circuit between two gates, we can not use it. But the poly layer can do that. Compared with the metal layer the poly layer has a very large resistance and consequently it will cause a large RC delay. On such occasions, we should move the gates between the two gates to somewhere else and place these two gates in the critical path as close as possible to each other so that they can be connected to the metal layer. It may result in more delay forthose gates that were displaced.

Distinguishing the fast gate and slow gate

Usually, for a logical gate with a number of fan-ins, the signal will not appear on the inputs simultaneously. Some signals appear earlier and some appear later. We can take a two-input NAND gate as an example, which is shown in figure 25. Note that there are two n-transistors connected together in series. Assume that both inputs are initialised to *0*. If input *in0* receives a logical *1* first, it will turn on the n-transistor connected to it. At this moment, the *in1* is still *0*, so no path to GND is developed and *out* is still *1*. At some moment, the *in1* receives a *1*, then its connected n-transistor is turned on. Since two n-transistors are all turned on, the *out* is discharged to *0*. For this case, the current should go through two n-transistors to discharge the *out* when the *in1* receives a *1*. If the input *in1* receives the *1* first, it turns on its connected n-transistor. Note that as soon as this transistor is turned on, it will discharge the point *a* to *0*. Thus, when the input *in0* receives a *1* and turns on the second n-transistor, the current only goes through one transistor to discharge the *out* node. It is certainly faster than the first case. Therefore, we define the *in0* as the fast gate, and the *in1* as the slow gate.

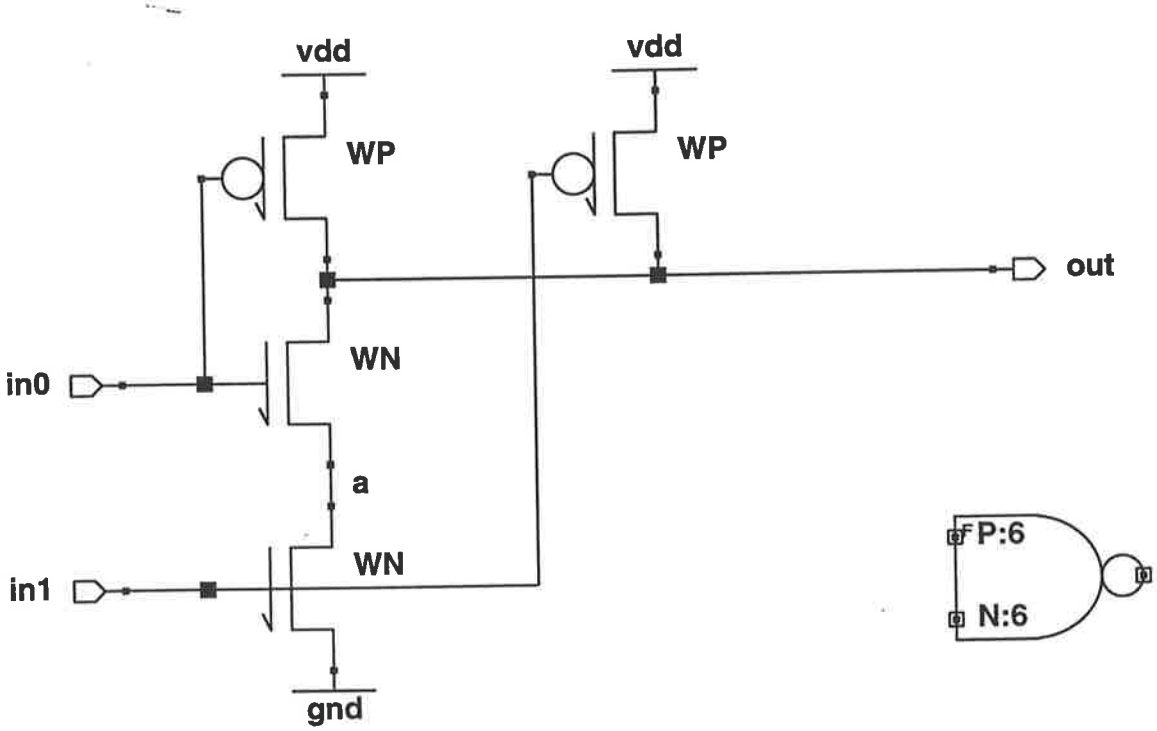

Figure 25 Circuit of an NAND gate

The signals in the critical path usually arrive later than the other signals in the same logic gate. Therefore, we always connect the signals in the critical path to the fast gates. Although the improvement from each gate is very small, it still contributes to reduce delay.

4.2.3.4 General Techniques to decrease delay

Some other techniques are widely used in the layout design of the output scheduler.

- **Carefully sizing the transistor**

As stated above, this output scheduler is designed manually. A significant advantage of custom design is that the size of the transistor can be selected flexibly according to its fan-out. Thus, we can use large transistors to drive large loads, so as to achieve high speed.

- **Stage Ratio**

In some case, the buffers have to drive a very large capacitance, such as a bus or an off-chip capacitive load. A small buffer will take more time to charge it, so a large buffer is needed. In order to achieve the best speed, we can use a chain of inverters where each successive inverter is made larger than the previous one until the last inverter in the chain drives the large load. The ratio by which each stage is increased in size is called the stage ratio. It has been shown that when the stage ratio is in the

PMOS transistor, its voltage rises from 0 to Vdd, and a certain amount of energy is drawn from the power supply. During the high-to-low transition, the capacitor is discharged, and the stored energy is dissipated in the NMOS transistor. We can compare it with dynamic logic. The power consumption in a dynamic network is solely determined by the signal-value probabilities, not by the transition probabilities. In other words, in the dynamic logic we should always precharge the gates no matter if there is a transition or not. Consequently, the dynamic logic will sink more power than the static CMOS[9].

4.2.4.2 Reducing the Effective Capacitance

The dynamic consumption of the static CMOS can be expressed with the following equation

$$p = cv^2 f$$

where c represents the load capacitance, v represents the voltage of the power supply and f means the frequency of a gate to be switched. For a particular technology, the voltage of the power supply is fixed and decreasing the voltage may affect the noise margin and affect the circuit speed. With the advance of technology, smaller propagation delays are becoming achievable. Consequently the switching frequency f is increasing. In order to reach very high speed, we can not reduce the frequency f to lower the power consumption. Therefore, the most effective way to reduce power dissipation is by reducing the capacitance.

- **Carefully sizing the transistors**

In the previous section we have discussed how the transistor size affects the switching speed. Since each switching operation of the combinational static CMOS is actually charging or discharging a capacitor, it is clear that the smaller the capacitor charged, the less power is consumed. Toward that end, each gate in the output scheduler is carefully sized to be as small as possible. When all the transistors are designed to be the minimum size, the power dissipation is minimised. However, that will affect the speed of the circuit. Therefore, when a gate is required to drive a large capacitance, the transistor must be sized up.

- **Carefully sizing the wires**

As we know, in CMOS technology, delay is basically caused by charging or discharging capacitance. In addition to the capacitance comes from transistors, long wire is another major source of capacitance. We should carefully size the width of long wires or wires connected to a large load. We seek to make the wire to be as narrow as possible. However, if the wire is connected to a large capacitor load or the wire itself is very long, we need a large buffer to drive it to achieve high speed. A large buffer implies a large current. The metal has a limitation on current density (usually it is 0.4 mA/um to 1.0 mA/um). If the current density of a current-carrying conductor exceeds the threshold value, the conductor atom will move in the direction of the current flow, and the conductor may eventually like a fuse. If we simply select a very wide metal as the wire, it will become a large capacitor and sink a lot of

power. Therefore, we should estimate the current needed for charging the capacitor and decide the width of the metal accordingly.

- **Avoid the extensive sharing of the bus**

Another approach to reducing the physical capacitance is to avoid the extensive sharing of the bus. In order to illustrate this point, let us analyse an example. Recall that in each row of elementary schedulers we use the discharged buses to realise the logical OR of the sixteen schedule arrays from elementary schedulers; and the final result is sent to the input status register. Clearly, this bus should be as long as the width of the output scheduler so as to connect all the sixteen elementary schedulers. No matter which elementary scheduler makes a schedule, it has to charge the whole bus so that the input status register can sense a *1*. The diagram is shown on the top of figure 26. To charge such a long bus it not only takes time, but also wastes power.

In order to avoid this drawback, we can divide the bus into two parts and each of them is connected to an OR gate (refer to the diagram in the bottom of figure 26). Each bus is still discharged to act as the logical OR of schedule arrays from its connected elementary scheduler. Indeed, we used this divided bus to realise a logical OR of the result of two buses that is the logical OR of 8 schedule arrays. Clearly, the divided buses have identical function to that of a signal bus, but each buffer should only drive a load half the load of the previous bus case. Therefore, this approach reduces the power dissipation.

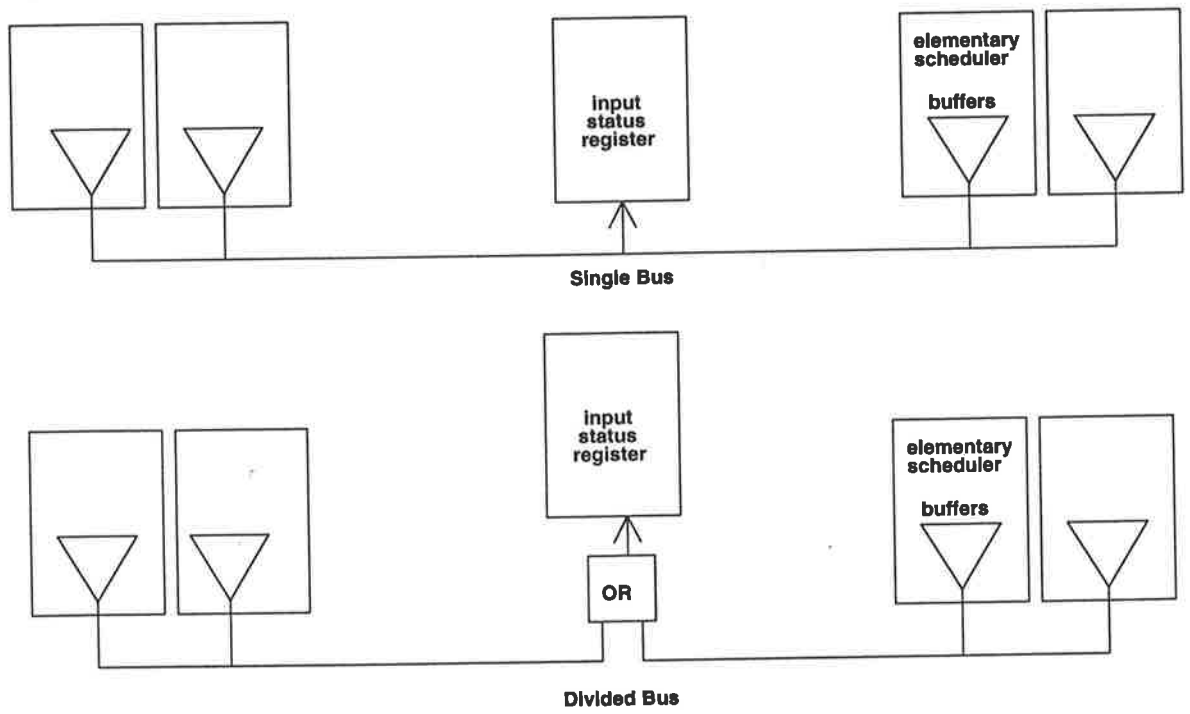

Figure 26 Avoid extensive bus sharing

4.2.5 Techniques for reducing the area

Usually, the area is the last factor to be considered for high performance circuit design and we often trade area for high speed and low power dissipation, but we can also reduce the area by careful layout. Clearly, optimally sizing the transistors is an effective way to achieve small area. In addition to that, some other techniques are used.

One approach is to carefully design the wires. In the modern CMOS technology multiple metal layers are employed so that the wires can be routed above the circuits, which helps to reduce the area. In very complex or very compact design, sometimes the available metal layers are not enough to route all the wires above the circuit; in this case, we have to place the wires in the spare places, which will take more area. In the layout design of the output scheduler, we need many wires to connect the subcells. For example, we need wires to connect the elementary schedulers in a column for the output status array; we need wires between the elementary schedulers and the input status register for priority information, input status array and scheduler array. In order to minimise the area, we hope to place these wires above the circuit layout. Towards that end, the width of the wires are calculated according to their current load, the space between wires is minimised and the placement and direction of the wires are well organised. All of these techniques ensure that most of the wires in the output scheduler are routed above the circuits.

Another approach is to divide the large buffer into a number of smaller parallel buffers. In the output scheduler, we often need some large buffers to drive the large

capacitive load. Referring to figure 22, we note that all of the layout is organised into many slices. The width of the slice may not be sufficient to place a very large buffer, so we divide the large buffer into a number of smaller buffers whose size is suitable for the height of the slice. Thus, no matter how large are the buffers, they can always be placed into the required dimension. Clearly, dividing large buffers into smaller ones can not reduce its absolute area, but it makes the subcell of the layout very regular. The cells with regular dimension and similar size are helpful to minimise the area.

4.2.6 I/O System Design

In this subsection, we will discuss another factor that significantly affects integrated circuit performance, I/O system.

Pads are the interfaces between the chip and the outside world. In the output scheduler four types of pads are used, namely, input pads, output pads, power pads and pad ring pads. The input and output pads are employed to exchange signals with other chips; the power pads are used to provide the power supply for the circuits; the pad ring pads are used to supply the power for the input and output pads.

Figure 27 shows the topology of the pads for the output scheduler. As we have mentioned above, there are 32 inputs and 32 outputs. Therefore, we needs 32 input pads and 32 output pads. On the top of the chip, there are 16 output pads for input addresses; the input pads for priority information and output addresses are placed on the left and right side of the chip respectively; the output pads for schedules are put

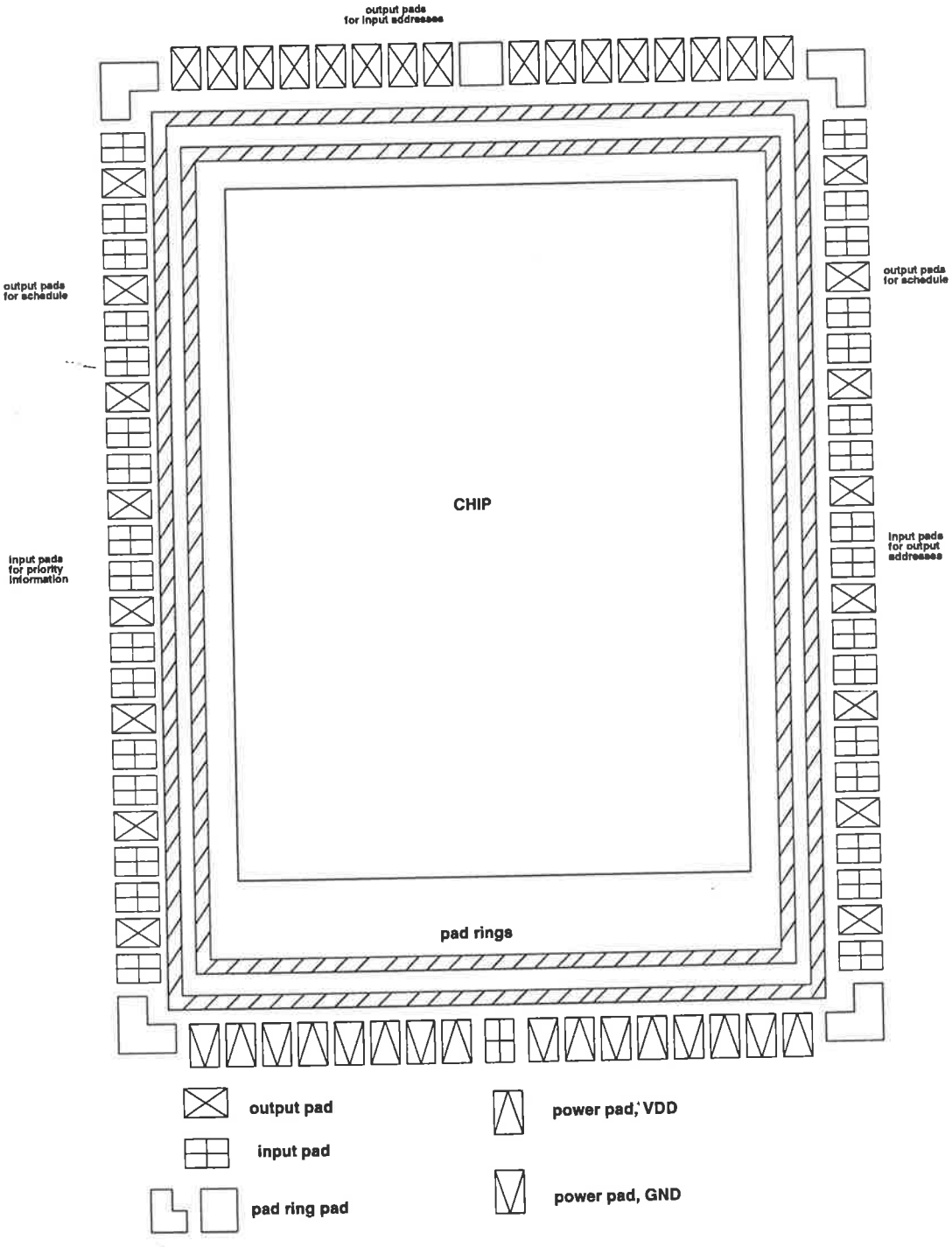

Figure 27 Placement of pads

evenly on the right and left side as well (see figure 27). In addition to those pads, an input pad is used for the clock input, which is placed at the bottom of the chip. There are eight power pads for VDD and eight power pads for GND. Moreover, we note that at each corner and in the centre of the top of the chip there are five pad ring pads. Each pad ring pad includes both the VDD pad and GND pad.

Note that between the circuits and the pads, there are two pad rings. These two rings are two metal rings to provide the power for all the input and output pads. One of the pad rings is connected to VDD, and the other is connected to GND. All the input and output pads are powerd from the pad rings. The pad rings are connected to the pad ring pads. The reason for using a separate power supply for the pads is to reduce the noise. As we know, the output pads are required to drive large capacitors, and provide high current dive for short periods. It may cause power bounce. If we use the same power supply as the circuits, these currents may flow through internal circuitry causing power and ground bounce. Moreover, note that one pad ring pad is put in the centre of the top of the chip. The output pads need a high current capability to drive the off chip capacitance. Typically, every pad ring pad used in this output scheduler can only provide enough power for eight output pads, so one pad ring pad is put there. On the left and right sides, there are only eight output pads and sixteen input pads that need only drive a small capacitance and sink little power, so the pad ring pads on the corner are enough to provide them power.

Note that we use eight pads for VDD and eight pads for GND. The number of the power pads is determined by two factors: the maximum current of the chip and the transient change of the current. Figure 28 shows the transient current curve of one

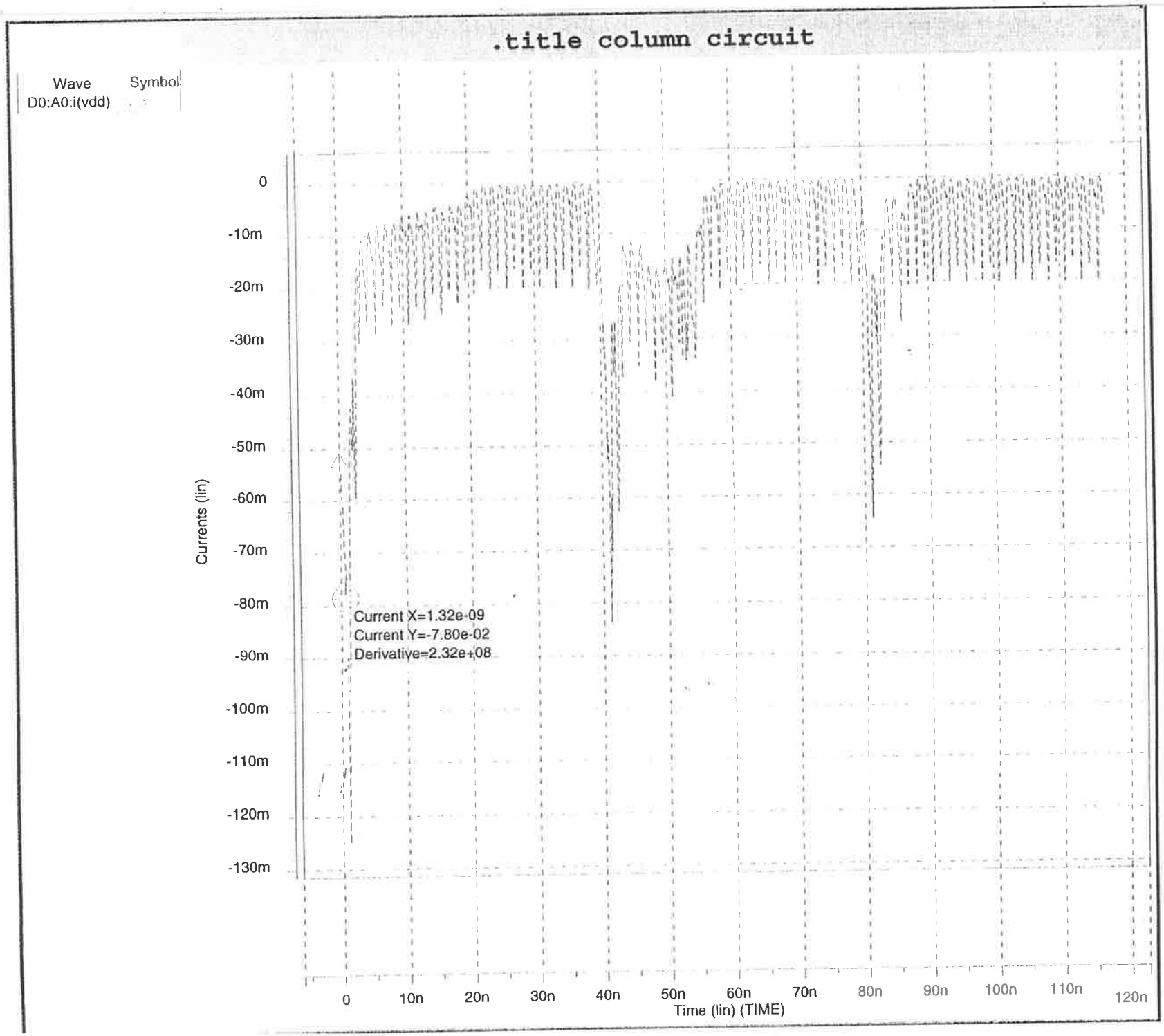

Figure 28
Transient currer waveform

column of elementary schedulers. Note that the maximum current is about 125mA. The overall circuit includes seventeen such units, so the overall maximum current drawn from VDD is about 2.125A. The maximum current that each power pad used here can provide is about 0.5A. Thus, we need only five power pads to provide sufficient power for the chip.

However, we note that such a large current is drawn from the power supply within a very short period. The change of voltage caused by the inductance of the bond wire is

$$DV = L\, di/dt$$

where *di/dt* is the rate of current change with respect to time. A rapid change of current results in a large voltage change possibly enough to cause the state of the circuits to be changed. In order to avoid that, we have to provide enough power pads to share the large current so that in each pad the *di/dt* can be reduced and the voltage change limited to a tolerable range.

The number of pad can be calculated according to the transient current wave in figure 28. The maximum *di/dt* for the simulated part of the circuit is about 2.3×10^8 A/s and a typical value for inductance of a bond wire is about 1nH. Thus, the voltage change can be estimated from the above equation as 0.23v. That means if we use one pad to provide the power to this part of the circuit, there is a maximum voltage swing of 0.23v. Since we are using a 2.5v power supply and complementary logic with a gate threshold around 1.25v, the permissible maximum ground an power bounce is 0.5v. Hence we can use a single pad to provide power for up to twice the amount of circuitry simulated in this example. The whole circuit is comprised of

seventeen such parts, so we use eight power pads to provide the power to the total circuit and so ensure that the ground power bounce is not more 0.5v. That is the reason why we use eight VDD pads and eight GND pads.

4.2.7 Power distribution

Finally, we will discuss power distribution in the output scheduler. In the design of the power distribution system of the output scheduler, two main factors are considered: one is IR voltage drop and the other is noise.

As shown above, the output scheduler requires high instantaneous power handling capability at the beginning of each time slot, so IR voltage drops along the power lines should be considered. The IR voltage drops degrade the circuit's noise margin and make the circuit less reliable. This becomes worse when the supply voltage is scaled down, because the magnitude of the voltage drops that can be tolerated are even smaller. An effective way to reduce the IR voltage drop is to reduce the resistance of the power line. In the design of output scheduler, we use the topmost and thickest metal level (Metal 5) which has a smaller sheet resistance compared with other metal layers to distribute the power. In addition, since this layer is used solely for power distribution, we can make it wide enough to reduce the resistance to an acceptable level.

Another factor that may affect the performance of the circuit is the noise on the power supply. We have illustrated that in order to reduce the noise on power supply we use eight power pads to share the high current. In addition, we use a separate

power supply for the pads so that no power or ground bounce caused by the pads can flow into the internal circuits.

As we have discussed, each input status register should shift the schedule out of the chip with a frequency of 400MHz, which is power hungry. In addition, each input status register should drive the output address, priority information and input status array into all the elementary scheduler in a row, so there are a number of large buffers in it. Therefore, the input status register will sink much more power than an elementary scheduler. Moreover, most of the power is drawn within a very short period, such as at the beginning of each time slot, so large power and ground bounce may happen. In order to reduce that, we provide sufficient capacitance between the power line and the substrate. When a power or ground bounce occurs, the connected bypass capacitor will be charged or discharged, which reduces the magnitude of power and ground bounce. The capacitors are placed close to the large buffers so that charging and discharging may be carried out effectively. Thus, the noise on the power supply is reduced significantly.

In this section, we mentioned that in order to improve the performance, many design parameters are selected on the basis of estimation. All the subcells of the circuits were simulated with Hspice to verify the correctness of estimation. If the estimated parameter is not good enough, it will be modified according to the simulation results.

4.3 The Simulation Result of the Output Scheduler

In the last few sections, we have addressed the techniques used to improve the performance of the chip. In this section, we give the simulation results for the circuit.

4.3.1 Delay and Power Dissipation

As we have discussed above, because the speed of a circuit is decided by its critical path, we should only simulate the delay of the critical path. According to the previous explanation, we note that the delay of the critical path is between the moment that the output status register sends the output status array to the moment that the updated output status array appears on the input of the output status register.

4.3.1.1 Simulation Environment

The delay of this critical path is simulated with Hspice. As we have mentioned, Hspice is suitable for accurate simulation of small circuits. We can not simulate this critical path in the environment of the whole output scheduler. Therefore, only a column of the elementary scheduler is simulated. Fortunately, it is enough to reflect the delay of the output scheduler. The entire power dissipation of the output scheduler can be estimated from the simulated part of the circuit.

Another point that should be mentioned is the selection of simulation parameters.

- **Simulation model**: As we know, the circuit will be fabricated on a wafer. The physical properties of the wafer may vary according to the position. Therefore,

we use a different model to simulate it. Basically, this consists of using models such astypical model, fast NMOS and fast PMOS, fast NMOS and slow PMOS, slow NMOS and fast PMOS, and finally slow NMOS and slow PMOS. Since we are simulating static digital circuit that is not very sensitive to the environment, the simulation is simply carried out with the typical model.

- **Temperature:** Usually, we have three choice for the temperature, namely 25°C, 75°C or 100°C, in which 25°C is the best case, 75°C is typical and 100°C is the worst cast. The scheduler is simulated in the environment of 100°C.
- **Processing Technology:** The technology for fabricating this scheduler is TSMC 0.25μm technology.

4.3.1.2 Selection of the Stimuli

Recall that for each elementary scheduler to work it needs such signals as the input status array, priority information, an enable signal and the output status array. Since our purpose of this simulation is to find out the maximum delay of the critical path, the stimuli must be selected carefully so as to achieve the worst delay of the scheduling.

The input status array and output status array are selected as "0000_0000_0000_0000". This value ensures that the elementary schedulers have sufficient scheduling resources. The priority information is set to "0000_0000_0000_0000" and all the 16 elementary schedulers in the column are enabled. These values make sure that each comparison unit in this column will take part in the scheduling. Therefore, we can achieve the worst delay from these stimuli.

4.3.1.3 Simulation Result

The delay of the critical path can be deduced from figure 29. The figure shows the two voltage curves: enable signal (dotted line) and 16th bit of the updated output status array (real line) that is the critical path delay. As shown in the figure, from 0ns to 40ns, the enable signal is *1*, that means in the first time slot all the elementary schedulers are enabled. The real line shows that the 16th bit of the updated output status array is turned to *1* after 20.8ns. That means the output scheduler can finish one time scheduling process within only half of the time needed to support 10Gb/s/Channel. This ensures that the scheduler can potentially support the switch matrix with even higher speed.

As stated above, the critical path delay consists of two parts: one comes from the blocking circuits in the elementary scheduler, the other is the delay between each elementary scheduler. From figure 29, we can find how much each delay is. At 40ns, the enable signal turns to *0*, which will turn off the elementary scheduler. Since the scheduler is turned off, the last bit of the output status array should be its previous value *0*, instead of being updated. Therefore, the 16th bit of the output status array will turn to *0* after the enable signal turns to *0*. The enable signal will go through all the blocking circuits in an elementary scheduler and finally reach the 16th unit. The figure shows that the delay between response of the 16th bit of the output status array and the change of the enable signal is 7.8ns, which is just the delay caused by the

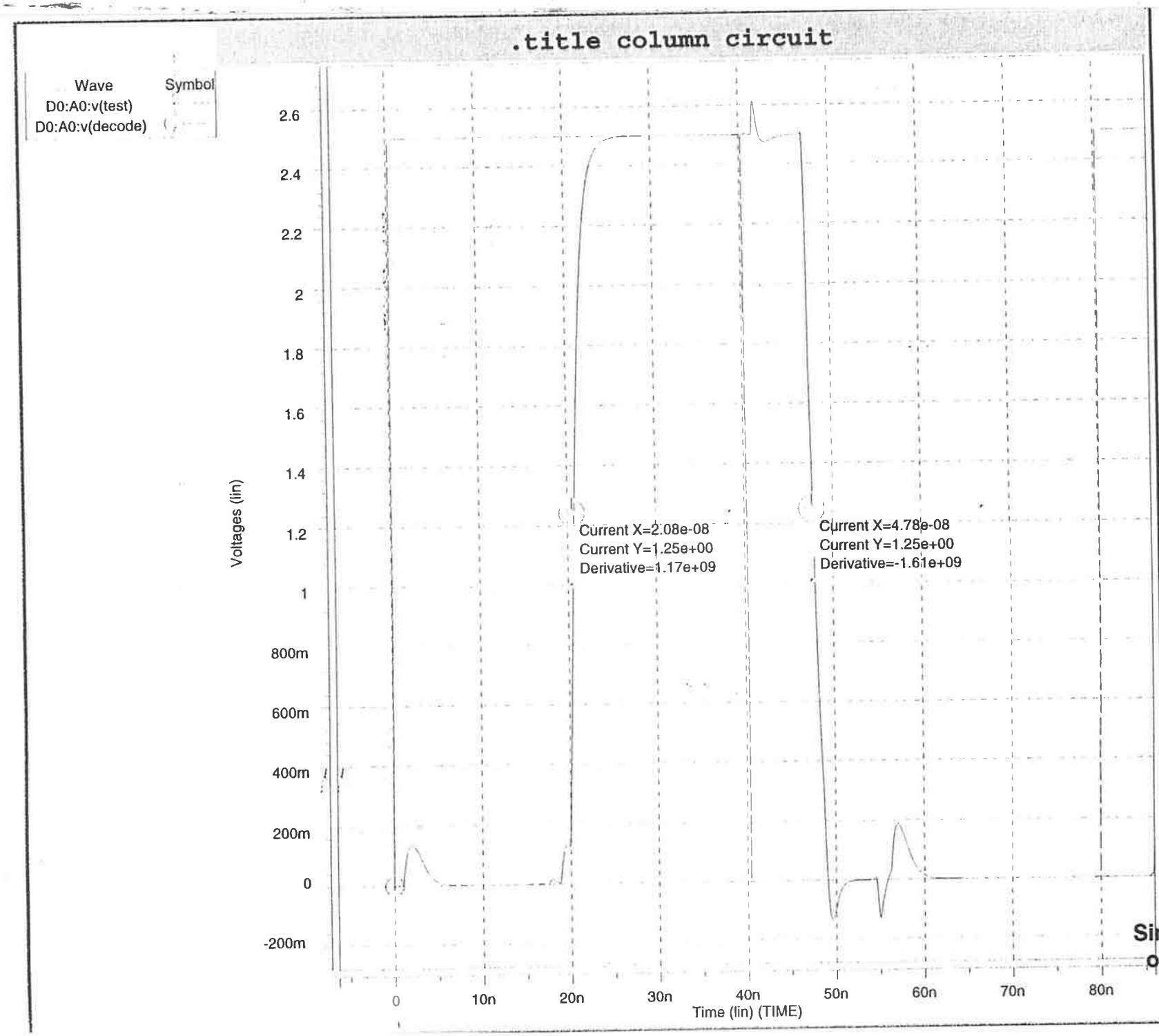

Figure 29
Simulation result of critical path

blocking circuit. The time that the output status array takes to ripple through all the 16 elementary schedulers should then be 13.0ns.

The overall power dissipation of the output scheduler can be achieved by the following equation:

$$P = I * V$$

Where I is the average current of the output scheduler and the V is the voltage of the power supply. The current is 314mA and V is equal to 2.5v. Therefore, the estimated power dissipation is 0.785w. The estimated power dissipation of the pads is about 0.1w. Clearly, this power dissipation is reasonably small, and it should not cause any trouble in packaging.

4.3.2 Size and Area

The whole output scheduler consists of about 600,000 transistors. The dimension of the circuit that does not include the pins is 2.81mm x 1.75mm. The area of the circuit is 4.9mm^2. Given the size of the circuit, this is a very compact design. The area of the chip that includes the pads can be calculated from figure26. According to current technology, the minimum centre to centre distance between each pad is about 150μm. Vertically, there are 27 pads and horizontally there are 20 pads. Thus, its dimension is 4.05mm x 3.00mm and the area is about 12.15 mm^2.

4.4 Summary

Chapter Five Discussion

In this section, we will discuss three approaches that can further improve the performance of this ATM switch.

5.1 Speed-up Two

At the architectural level, Sarkies and Main [6] showed that the switch performance is improved if the switch has an internal speed-up of two. With a speed-up of two, each time slot can be scheduled with two cells and two cells are switched out simultaneously through the switch. This makes the switch work more like an ideal output buffered switch with all its advantages.

The diagram of an ATM switch with a speed-up of two is shown in Figure 30. Note that each input port controller (IPC) sends two cells to the switch matrix at the same time. Therefore, a 16 x 16 switch matrix should be used for a 8 x 8 ATM switch. Similar to the switch without speed-up two, each input port controller sends an output address and priority information to the output scheduler. However, since two cell can be scheduled in one time slot, the scheduler needs two schedules to describe the scheduling results for each input port controller (see figure 30). In addition, the output schedule generate eight pairs of output addresses for the switch matrix.

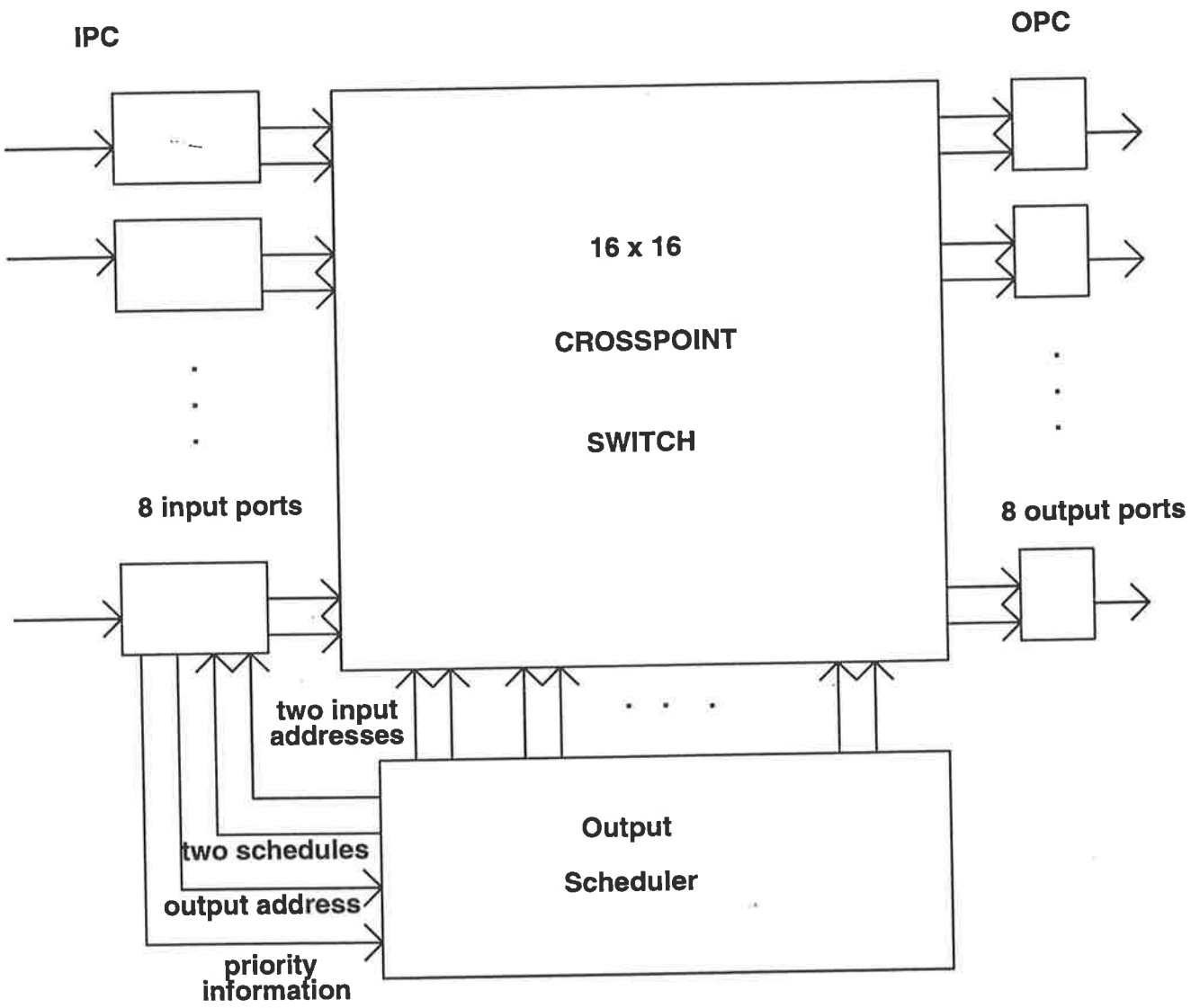

Figure 30 Architecture of an input buffered ATM switch with speed-up two

In order to support the speed-up, the internal structure of the output scheduler should also be changed accordingly. Since speed-up two permits two cells to be scheduled into one time slot, we need two input status arrays (A and B) and two output status arrays (A and B) to describe the states of the input and output ports. Clearly, for each time slot there are four possible combinations of scheduling status, namely AA, AB, BA, BB. Therefore, we need four elementary schedulers for each input-output pair to make a schedule for it. We can define it as an elementary scheduler group. A diagram of an elementary scheduler group and its corresponding input and output status registers are shown in figure 31. The comparison unit and the schedule register in each elementary scheduler are identical to that described in the previous chapters. Each elementary scheduler in a group still receives the input status array directly from the input status register. Each elementary scheduler also receives the output status array from its upper elementary scheduler. The input status register collects a logical OR of the schedule arrays from each elementary scheduler.

The difference is that each elementary scheduler employs a signal to interface with other elementary scheduler in the group (see figure 31). This signal is used to prevent that one cell being scheduled into the same time slot multiple times. For example, if the input status and output status are all *00*, both elementary scheduler AB and AA may make a schedule. If this happens, the cell is scheduled twice into the same time slot. It wastes the scheduling resource and may cause a problem in switching. Therefore, we need some interface signals to make sure that only one of the four elementary schedulers is enabled in each time slot. In other words, the scheduling process in a scheduler group should be carried out one by one. The interface signal is generated by the logical OR of each bit of schedule results. In other words, if a

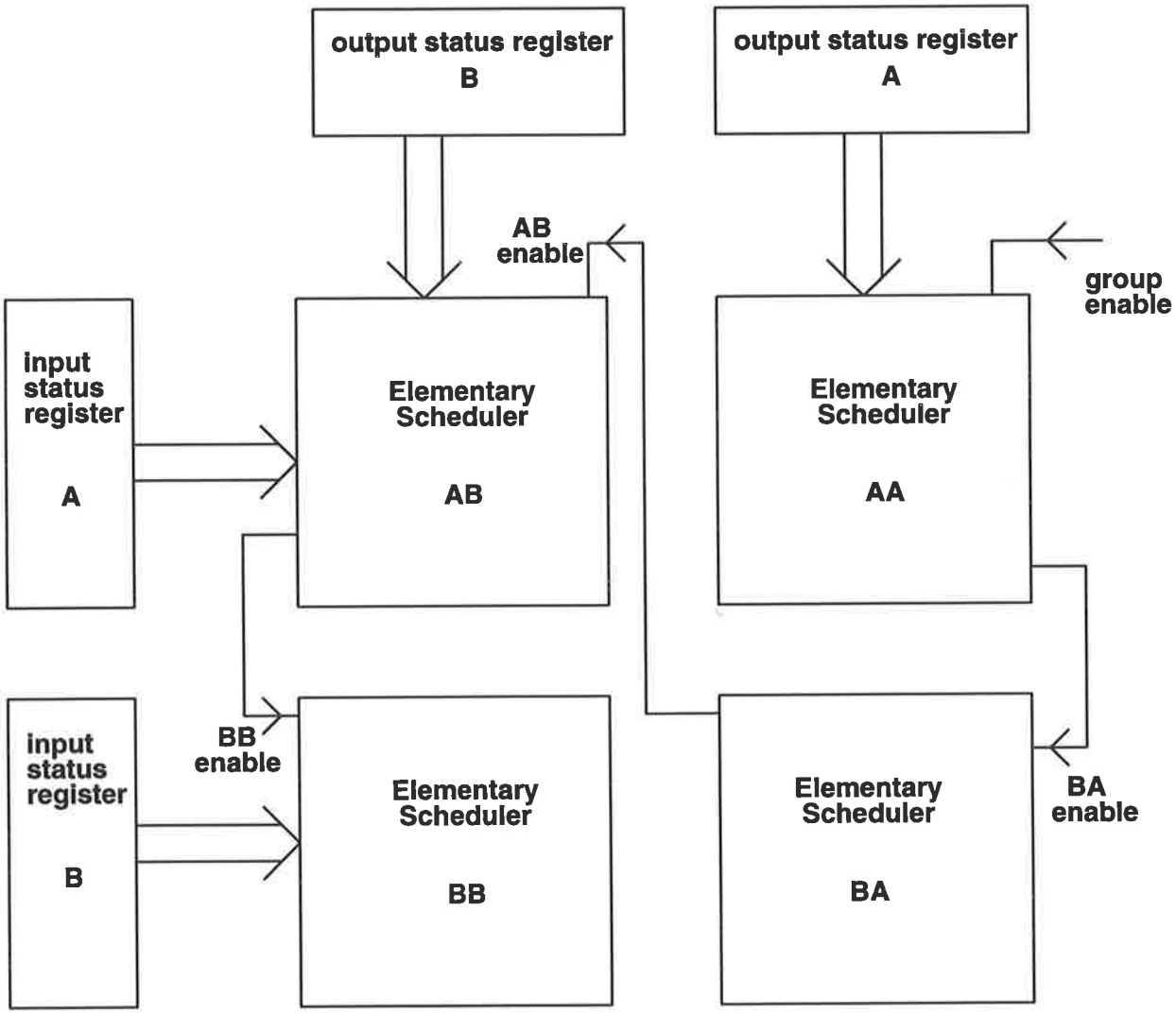

Figure 31 Diagram for an elementary scheduler group

schedule is found in an elementary scheduler, it will disable all its subsequent elementary schedulers. If not, it will inform its subsequent elementary scheduler to go on scheduling. Such a sequence is arbitrarily selected: AA, BA, AB and BB. The enable signal of elementary scheduler AA is connected to the group enable. That means, if this group is selected, the elementary scheduler AA will be enabled and the status in input status register A and output status register A will be compared within elementary scheduler AA. If no schedule is made in AA, BA is enabled, then AB is turned on and so on. If a schedule is made in AA, all the subsequent elementary schedulers will remain disabled.

Besides the addition of the interface signals, another difference from the scheduler without speed-up two is that only the first row of elementary schedulers in each group can receive the output status array directly from the output status register, which is used to defeat unfairness.

Moreover, due to speed-up two, an output scheduler with an array of 16x16 elementary schedulers, 16 input status registers and 16 output status registers becomes an output scheduler that has only 8 x 8 elementary scheduler groups, with eight input and output status register pairs. Therefore, such an output scheduler can only support an 8x8 ATM switch.

5.2 A Possible Way to Improve the Speed

Although the current circuit is fast enough to satisfy the design objective, we still hope it to be faster. The speed can be further improved when we use some lookahead

in the elementary scheduler. From the discussion in chapter two, we note that each comparison unit only interfaces with its immediately subsequent unit and the interface signal ripples through all the 16 units one by one. In other words, each unit only passes its scheduling result to its immediately subsequent unit. Referring to figure 17, we note that each blocking circuit results in two gate-delays. That means the overall delay from the blocking circuit is thirty gate-delays.

If we use some lookahead among the comparison units, the delay caused by the blocking circuits can be reduced. In this case, each comparison unit should interface with all its subsequent units, instead of its immediately subsequent one. In other words, each unit informs all its subsequent units of the comparison result. A diagram that illustrates this approach is shown in the figure 32. Note that the first comparison unit sends the interface signals to all of the subsequent 15 units, unit2 sends the interface signal to its subsequent 14 units and so on. Part of the circuits with lookahead is shown at the bottom of the figure. The figure shows the circuits of four comparison units. Note that the circuits of unit1 are identical to that without lookahead. The difference is that its interface signal (*down1*) is sent to all its fifteen subsequent units. Similarly, the unit2 sends its interface signal (*down2*) to all its fourteen units. Now let us study how each unit interfaces with its preceding units. Since unit2 should only interface with one preceding unit, unit1, its interface signal, *up2,* is connected to the signal *down1* directly. Unit3 should receive the interface signal from both unit1 and unit2. Therefore, its interface signal *up3* receives the logical OR of *down1* and *down2.* For the same reason, unit16 should use a 15-input OR gate to interface with all its preceding units (see figure 32). From the circuit

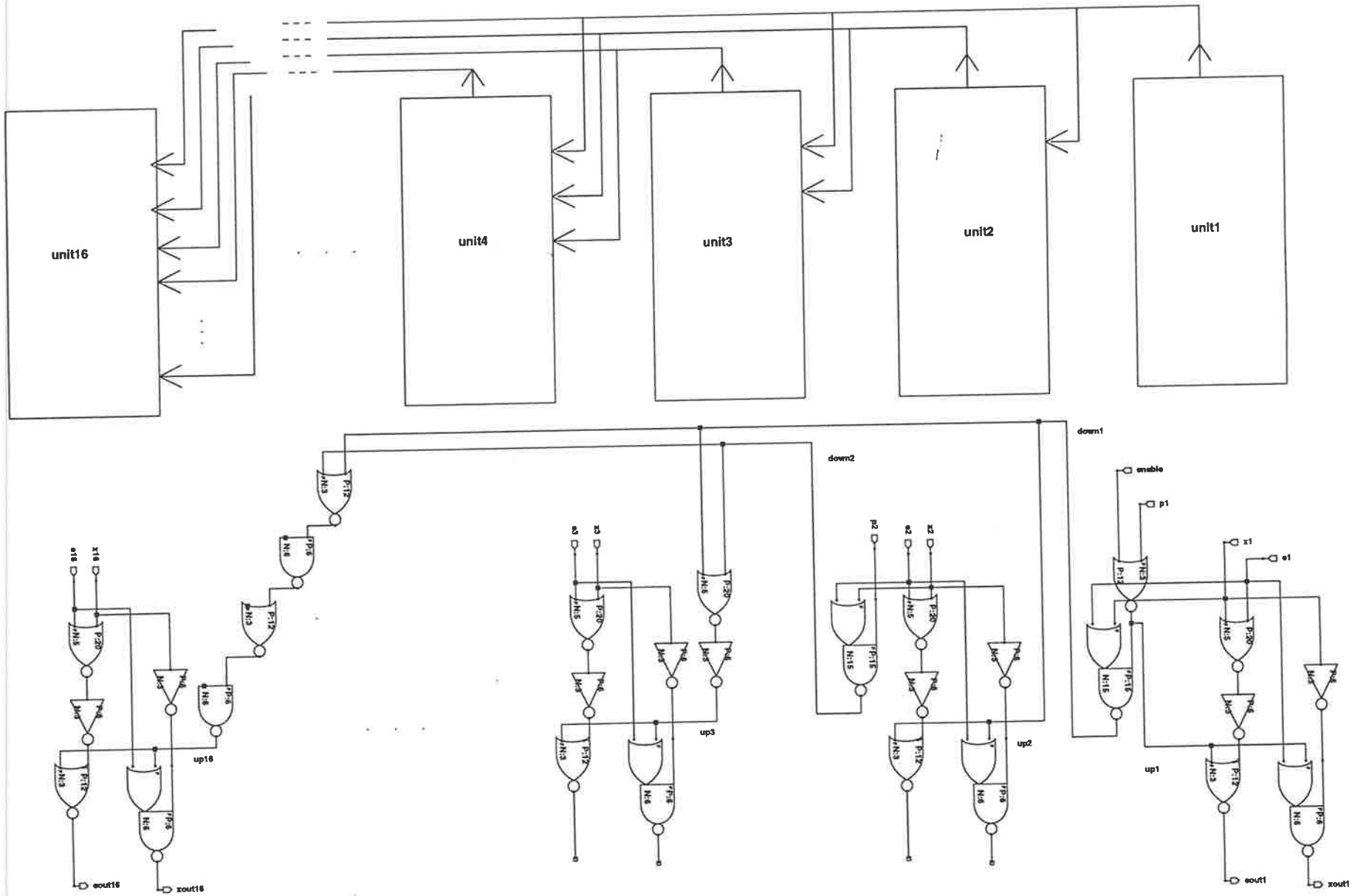

Figure32 Structure and circuit of elementary scheduler with lookahead

shown in figure 32, we can estimate the delay for this approach. The critical path should begin at the *enable* signal and end at *x16*. Note that this path includes seven gates. According to our previous discussion, we know that there are about 30 gate-delays introduced by the blocking circuits. Clearly, lookahead can reduce delay from the blocking circuits significantly.

Nevertheless, what we pay for the high speed is greater area and design effort. With the lookahead, each unit should interface with a few units, so each unit should add an OR gate to collect the interface signals. This takes more room. Since the output scheduler includes 256 elementary schedulers, any addition of each elementary scheduler's area will affect the whole circuit very much. Moreover, we note that each unit's circuit will be somewhat different, which implies that each unit should be designed separately. In addition, lookahead results in a more complicated layout design for the comparison unit.

In the design of this output scheduler, we have comfortably satisfied the requirements of the speed, so we did not employ lookahead. However, if the speed requirement is very strict, it is worthwhile to trade area, power dissipation and other factors for speed.

5.3 Challenges on packaging

Packaging is as important as, and often even more critical than, transistors in determining the overall performance of a system. As we have mentioned above, the output scheduler should exchange the information with other chips at very high

frequency. In order to support such high frequency, packaging technology should be selected carefully.

In traditional packaging technologies, the circuit is fabricated on a silicon die and the die is put into a chip carrier, then the chip carrier is placed on a print circuit board and interfaces with other chips. [10]. The large parasitic inductance of the bonding wires and the transmission lines will cause noise on the signals. The reason can also be illustrated from the following equation:

$$dV = L\, di/dt$$

The current used to charge and discharge bond wire and transmission line may change with high frequency, so di/dt would be substantially high. Due to the large inductance, L, from the bonding wires and transmission lines, the signal voltage signal will be changed and this is a source of noise. On the other hand, the capacitance of the bond wire and transmission line is quite large. The output pads take more time to charge or discharge such a large capacitor to exchange information with other chips. Therefore, it increases the delay of the chip and limits the highest frequency that the pads can reach.

In order to reduce the noise on the signal and improve the speed, an effective way is to minimise the inductance and capacitance from the bonding wire and transmission lines. Multichip Module (MCM) technology is a solution. In MCM, a number of chips are placed on one substrate, which provides smaller inductance and capacitance electrical connections among the dice than that provided by traditional single-chip carriers and PCB.

There are a number of alternatives of MCM. The one that is appropriate for packaging this ATM switch is silicon-on-silicon hybrid [10]. A silicon substrate is used as an interconnection medium to hold multiple chips. Thin film interconnections are fabricated on a wafer, and separately processed dice are mounted on this silicon substrate. A significant advantage is that chips fabricated in different technologies (CMOS, bipolar or GaAs) can be placed on the same hybrid package. The silicon substrate can also potentially contain active devices that serve as chip-to-chip driver, and bus and I/O multiplexers.

The ATM switch discussed in this thesis may contain chips fabricated with different technologies. For example, the scheduler is designed with CMOS and the switch matrix is most likely fabricated with GaAs. This is one reason that this packaging technology is suitable for this ATM switch.

Here we only present a basic idea on the selection of packaging technology, there are many topics should be further researched to make each chip communicates with other chips perfectly at high frequency.

5.4 Summary

In this chapter we discussed three topics: speed-up two, lookahead, and packaging. We discussed the speed-up two and the corresponding modification of the scheduler's structure to support speed-up two. Then, we discussed a possible way to reduce the delay caused by the blocking circuits. Its drawback is also analysed. The

packaging is a critical part that will affect the performance of the chip. Since the output scheduler works at a very high speed, a high quality package is necessary to ensure the high performance of the chip. A possible Multi-chip Module technology is discussed in this chapter. We mentioned some advantage over conventional packaging technologies.

Chapter Six Conclusion

In this thesis, we discussed the design of an input-buffered high-performance ATM switching system, which employs a time scheduling algorithm developed by Sarkies and Main [6]. This research has achieved the following design objectives:

- the architectural of an input-buffered ATM switch. The switch includes four major parts, the input port controller, switch matrix, scheduler and output port controller. The scheduler is a key part of this project, so it is designed to the circuit level. The input port controller and switch matrix that interface with the scheduler is designed to the architecture level. We demonstrated that each part of the switch can coordinate with others perfectly.

- demonstrate that the input port controller can not only realise such basic functions as generating the scheduling request and processing the scheduling result, but also offer some advanced functions such as variable priority threshold and multicasting that is an enhancement to the algorithm.

- According to the time scheduling algorithm and the design requirement of this ATM switch, the architecture of the output scheduler is designed. Subsequently, we design the VHDL models, schematics and layout of the scheduler.

- The simulation results of the layout show that the maximum delay of a scheduling process is about 21ns. Clearly, it is fast enough to satisfy our design

objective, 40ns. The whole output scheduler includes 600,000 transistors. The estimated power dissipation of the circuitry is about 0.785 watts and the power dissipation for the pads is about 0.1w. Given such a big circuit that works at very high speed, this power dissipation is reasonably low. The dimension of the circuit that doesn't include the pads is 4.9mm^2. The overall area that includes the pads is about 12mm^2.

- Finally, we discussed the speed-up two that improves the performance of the algorithm and the corresponding modification of the structure due to the speed-up two is analysed; the circuit with lookahead is discussed to improve the speed of the output schedule; a possible way to package the circuit is discussed;

The design satisfies all the objective of the project and the circuit of the output scheduler can potentially support even an higher speed switch matrix.

Reference:

1. F.Halsall, "Data communication, Computer Networks and Open systems": ADDISON-WESLEY, 1992.

2. L.G.Cuthbert, "ATM: the broadband telecommunication solution": Institute of Electrical Engineering, 1990.

3. R.Handel, M.N. Huber, "Integrated Broadband networks": ADDISON-WESLEY, 1991.

4. K.S.Lowe, "A GaAs HBT 16x16 bit 10-Gb/s/sChannel Crosspoint Switch", In IEEE Journal of Solid-State Circuits, VLO32, No.8, August 1997.

5. R.Savara, A.Turudic, "A 2.5 Gb/s 16 x 16 Bit Crosspoint Switch with Fast Programming", IEEE GaAs IC Symposium, 1995, pp. 47-48.

6. J.Main and K.Sarkies, "Cell Scheduling Using Status Arrays in Input Buffered ATM Switches", IEEE BSS'95, Poznan, April 19-21, 1995, pp.333-339.

7. N.McKeown, M.Izzard, "The Tiny Tera: A packet Switch Core" IEEE Micro pp 26-33 January 1996.

8. N.H.E. Weste, K.Eshraghian, "Principles of CMOS VLSI Design: A systems Perspective", ADDISON WESLEY, 1993.

9. J. M. Rabaey, "Digital Integrated Circuits: A Design Perspective", PRENTICE HALL, 1996.

10. H.B.Bakoglu, "Circuits, Interconnections, and Packaging for VLSI", ADDISON-WESLEY, 1990.